Estimating Story Points from Issue Reports

Simone Porru
University of Cagliari
Cagliari, Italy 09123
simone.porru@diee.unica.it

Alessandro Murgia
University of Antwerpen
Antwerpen, Belgium 2020
alessandro.murgia@uantwerpen.be

Serge Demeyer
University of Antwerpen
Antwerpen, Belgium 2020
serge.demeyer@uantwerpen.be

Michele Marchesi
University of Cagliari
Cagliari, Italy 09123
michele@diee.unica.it

Roberto Tonelli
University of Cagliari
Cagliari, Italy 09123
roberto.tonelli@diee.unica.it

ABSTRACT

Estimating the effort of software engineering tasks is notoriously hard but essential for project planning. The agile community often adopts issue reports to describe tasks, and story points to estimate task effort. In this paper, we propose a machine learning classifier for estimating the story points required to address an issue. Through empirical evaluation on one industrial project and eight open source projects, we demonstrate that such classifier is feasible. We show that —after an initial training on over 300 issue reports— the classifier estimates a new issue in less than 15 seconds with a mean magnitude of relative error between 0.16 and 0.61. In addition, issue type, summary, description, and related components prove to be project dependent features pivotal for story point estimation.

CCS Concepts

•**Software and its engineering** → **Agile software development;** Software evolution; Maintaining software;

Keywords

Machine learning; story points; issue report; agile

1. INTRODUCTION

Since the publication of the agile manifesto many organizations have chosen to adopt agile methods to drive software development [8]. In the agile context, effort estimation is pivotal for effective project management. Indeed, good estimates are essential to avoid poor resource allocation [9, 25].

A common means to estimate task effort are the story points [32, 11]. Within the context of agile development, story points are typically assigned via structured group meetings named *Planning Poker* sessions [14]. These meetings heavily rely upon human judgement: the better the developers understand the task, the better they can provide a sound estimate. However, human judgement may be hindered by several factors. Humans are generally optimistic and this bias is even more evident in group interactions [10, 5, 6]. In addition, developer estimation is known to be influenced by the presence of a project manager, other senior developers, or dominant personalities in the meeting [4].

To address these problems, we explore the use of a machine to support story point estimation. More specifically, we propose a machine learning classifier which leverages information within an issue report to file it in the most appropriate category. We target issue reports since they are commonly used by developers to describe development tasks, and are also heavily used in Planning Poker sessions [26].

The advantage of employing an automated classifier is threefold. First, the classifier has detailed knowledge about the project since its inception and produces its estimates on the basis of all the previous issues in the issue tracking system. Second, the classifier does not feel any pressure from or is influenced by other individuals, since its estimates can be traced back to the features used for classification. Third, the estimation is repeatable and predictable: the machine will never get tired and will always provide a consistent output.

We envision this classifier working *within* the team, providing its estimates along with other developers. In this manner, the classifier plays the role of an extra developer that produces an estimate in real-time on the basis of an objective analysis. As a first step towards building this classifier, we investigate whether it satisfies the preconditions to be successfully integrated during project planning. For this reason, we pursue the following research questions:

RQ1: What performance can be expected from a machine learning classifier when estimating story points? We aim to provide real-time support to a development team in story point estimation. Therefore, we need to state upfront how fast the machine can provide an answer, and how trustworthy such answer is.

RQ2: Which attributes of an issue report are relevant for estimating story points? To gain credibility within a development team, the classifier should be able to provide the rationale behind the estimates it produces. As part of the answer, we investigate the dominant features employed by the classifier.

RQ3: How much training data is needed to obtain

PROMISE 2016, September 09 2016, Ciudad Real, Spain

DOI: http://dx.doi.org/10.1145/2972958.2972959

a satisfactory estimation? Agile teams work in small sprints, hence the production of a few hundred issues over the course of a project. Since the correctness of the classifier estimation depends on the amount of training data, we investigate how the behavior of the classifier changes when we increase the amount of training data and from which point onwards the estimates become satisfactory for the team.

To address these research questions, we rely on a data set containing issue reports stored in the JIRA repositories obtained from one industrial and eight open source projects. To evaluate the quality of the estimation we use the Mean Magnitude of Relative Error (MMRE), a cost estimation metric often used while investigating effort estimation in agile contexts [30]. This way, we demonstrate that it is feasible to exploit the information available in issue reports to estimate story points in real-time. Indeed, for all projects but one, the classifier achieves a MMRE spanning from 0.16 to 0.61 after an initial training on at least 300 issues. Moreover, once the system is set up, the time required for estimating a new issue ranges from 8 to 15 seconds.

The rest of the paper is organized as follows: Section 2 gives background notions on story points and issue tracking systems; Section 3 presents the issue selection criteria, the preprocessing, the feature extraction and selection, and the validation process; Section 4 addresses the research questions; Section 5 discusses threats to validity; Section 6 presents related work and, finally, Section 7 draws the conclusions and presents future work.

2. BACKGROUND

This section describes what story points are and how they are estimated in agile development. In addition, as the issues were retrieved from JIRA, this issue tracking system is also introduced, along with the concept of issue.

2.1 Story Points

The story point is a metric used by agile teams to estimate the effort required to complete a development task [32][11]. Rather than quantifying the amount of work that needs to be done to complete a given task, the number of story points is an estimate of how hard a given task is for the development team. As a first step, the team usually agrees on the number of story points that a baseline task deserves. From that point on, effort estimation is based on the comparison with that baseline. Story points are commonly assigned following a sequence which slightly deviates from the Fibonacci series (i.e. 0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞) [13]. This sequence reflects the uncertainty inherent to the estimation of complex tasks in real software projects [11]. Note that the story points are meant to represent consensus within a team and not an objective estimate of the task effort. In fact, different teams may, for the same task, provide different estimates.

2.2 Effort Estimation – Planning Poker

Most software projects rely solely on human judgment for effort estimation [29]. Among effort estimation techniques based on human judgment, those based on group estimation are the most popular. More precisely, when it comes to story points estimation, the most common approach is the Planning Poker [14]. Planning Poker requires the customer to discuss an issue they wish to be addressed. Then, developers meet for a poker session where, for each issue to be estimated, participants individually select a card with the chosen story points, and finally reveal all the cards at the same time. The developer that provide the lowest and highest estimate must justify their choice, thus eventually triggering further discussion which is followed by another group estimation. The process continues until the team agrees upon a consensus estimate. Note that the story points in our industrial dataset stem from Planning Poker sessions; for the open source projects we do not know whether Planning Poker was adopted.

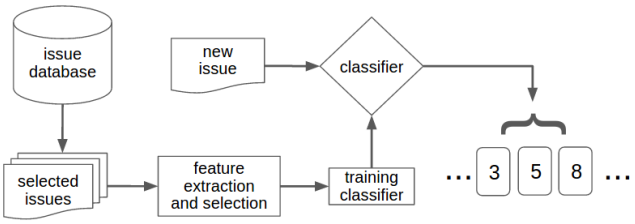

Figure 1: Estimation flow chart

2.3 JIRA

JIRA is a widely known issue tracking system. Here, an issue may represent a software bug, a project task, an enhancement, or anything that requires maintenance activity from the developers [17, 16]. In JIRA an issue is characterized by several standard fields and, quite often, also by fields introduced by JIRA administrators, known as custom fields. Among standard fields, we note: the *Summary* and the *Description*, which report textual information about the issue; the *Assignee*, which refers to the developer appointed to work on the issue; the *Component/s*, that specifies in which module of the project the task has to be implemented; the *Issue Type* (e.g. `Bug`, `New Feature`), which clarifies the required maintenance activity; the *Status* (e.g. `Open`, `Resolved`, `Closed`), which keeps track of the issue life cycle. Agile teams also use the *Story Points* custom field to record the estimated issue effort.

In addition to creating custom fields, JIRA administrators are also allowed to customize field values to satisfy their organization guidelines. This often happens with the *Status* field. In particular, an issue can be characterized by a *Status* marked with values such as `Closed`, `Fixed`, `Completed` or the like, which ultimately have the same meaning. Also the *Resolution* field can be set to different values that share the same meaning, such as `Completed` and `Fixed`. In fact, different organizations generally use a different *Status*/*Resolution* combination to refer to the same stage of the issue life cycle.

3. EXPERIMENTAL SETUP

This section presents the cases under investigation and the construction of a data set for training and testing the classifier. Then, it describes the approach used for issue selection, feature extraction, and feature selection. All the previous activities are reported in Figure 1, for the case when the classifier is used in Planning Poker sessions.

3.1 Cases

The dataset is composed of both industrial and open source issue reports mined from JIRA repositories. The dataset in-

Table 1: Issue types per project (number of issues (percentage))

Project	Total	Bug	Documentation	Epic	Improvement	New Feature	Story	Sub-task	Task	Technical Debt	Technical task	Wish
APSTUD	228	129 (56.58%)	0 (0.0%)	0 (0.0%)	39 (17.11%)	0 (0.0%)	46 (20.18%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	14 (6.14%)	0 (0.0%)
DNN	858	551 (64.22%)	0 (0.0%)	0 (0.0%)	157 (18.3%)	4 (0.47%)	73 (8.51%)	48 (5.59%)	25 (2.91%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
MESOS	387	169 (43.67%)	26 (6.72%)	0 (0.0%)	101 (26.1%)	0 (0.0%)	6 (1.55%)	0 (0.0%)	84 (21.71%)	0 (0.0%)	0 (0.0%)	1 (0.26%)
MULE	805	440 (54.66%)	0 (0.0%)	0 (0.0%)	153 (19.01%)	67 (8.32%)	9 (1.12%)	1 (0.12%)	135 (16.77%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
NEXUS	421	332 (78.86%)	0 (0.0%)	0 (0.0%)	78 (18.53%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	10 (2.38%)	1 (0.24%)	0 (0.0%)	0 (0.0%)
SCR	**699**	**469 (67.1%)**	**0 (0.0%)**	**0 (0.0%)**	**0 (0.0%)**	**0 (0.0%)**	**36 (5.15%)**	**0 (0.0%)**	**194 (27.75%)**	**0 (0.0%)**	**0 (0.0%)**	**0 (0.0%)**
TIMOB	634	524 (82.65%)	0 (0.0%)	3 (0.47%)	52 (8.2%)	36 (5.68%)	17 (2.68%)	1 (0.16%)	0 (0.0%)	0 (0.0%)	1 (0.16%)	0 (0.0%)
TISTUD	1215	751 (61.81%)	0 (0.0%)	0 (0.0%)	200 (16.46%)	7 (0.58%)	196 (16.13%)	1 (0.08%)	0 (0.0%)	0 (0.0%)	60 (4.94%)	0 (0.0%)
XD	360	87 (24.17%)	0 (0.0%)	0 (0.0%)	34 (9.44%)	0 (0.0%)	231 (64.17%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	8 (2.22%)	0 (0.0%)
all	*5607*	*3452 (61.57%)*	*26 (0.46%)*	*3 (0.05%)*	*814 (14.52%)*	*114 (2.03%)*	*614 (10.95%)*	*51 (0.91%)*	*448 (7.99%)*	*1 (0.02%)*	*83 (1.48%)*	*1 (0.02%)*

Table 2: Story points developers estimates per project (number of issues (percentage))

Project	Total	<1	1	2	3	5	8	13	20	40
APSTUD	228	1 (0.44%)	21 (9.21%)	7 (3.07%)	20 (8.77%)	62 (27.19%)	70 (30.7%)	40 (17.54%)	5 (2.19%)	2 (0.88%)
DNN	858	3 (0.35%)	375 (43.71%)	295 (34.38%)	130 (15.15%)	41 (4.78%)	13 (1.52%)	1 (0.12%)	0 (0.0%)	0 (0.0%)
MESOS	387	3 (0.78%)	120 (31.01%)	98 (25.32%)	106 (27.39%)	45 (11.63%)	14 (3.62%)	1 (0.26%)	0 (0.0%)	0 (0.0%)
MULE	805	171 (21.24%)	152 (18.88%)	45 (5.59%)	113 (14.04%)	154 (19.13%)	138 (17.14%)	32 (3.98%)	0 (0.0%)	0 (0.0%)
NEXUS	421	161 (38.24%)	199 (47.27%)	44 (10.45%)	17 (4.04%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
SCR	**699**	**89 (12.73%)**	**174 (24.89%)**	**187 (26.75%)**	**157 (22.46%)**	**72 (10.3%)**	**18 (2.58%)**	**2 (0.29%)**	**0 (0.0%)**	**0 (0.0%)**
TIMOB	634	13 (2.05%)	47 (7.41%)	63 (9.94%)	182 (28.71%)	192 (30.28%)	102 (16.09%)	35 (5.52%)	0 (0.0%)	0 (0.0%)
TISTUD	1215	1 (0.08%)	50 (4.12%)	64 (5.27%)	303 (24.94%)	489 (40.25%)	262 (21.56%)	45 (3.7%)	1 (0.08%)	0 (0.0%)
XD	360	1 (0.28%)	117 (32.5%)	76 (21.11%)	88 (24.44%)	54 (15.0%)	24 (6.67%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
all	*5607*	*443 (7.9%)*	*1255 (22.38%)*	*879 (15.68%)*	*1116 (19.9%)*	*1109 (19.78%)*	*641 (11.43%)*	*156 (2.78%)*	*6 (0.11%)*	*2 (0.04%)*

cludes different issue types, all with story points, as shown in Table 1. Even if the terms used for issue types depends on the project organization, the type `Bug` is present in every case study and is the most represented type, accounting for more than 60% of the issues in the dataset. The distribution of story points per class is reported in Table 2. Here we can see that 89% (5000/5607) of the issues fall within classes ranging from 1 to 8 story points.

Industrial Project

We use 699 issues belonging to the issue tracking system of Inventive Designers, a Belgian software company. The main product-suite, Scriptura Engage, offers solutions to create and manage customer communications for print, email, online, social, mobile, or interactive tasks. For the experiment, one of the development teams was involved. This team is composed of 5 developers and adopts a customized version of the Extreme Programming (XP) process. The team works on issue reports, which describe the smallest units of work. In the last two years they adopted Planning Poker. On these data, we perform a retrospective analysis, namely the data recorded is not influenced by our investigation.

We perform a retrospective analysis on these data, namely, the recorded data is not influenced by our investigation.

Open Source Projects

To allow for comparison against the industrial project, we selected a range of open source projects that use JIRA, provide public access to the repository, and record estimated story points. These projects provide a wide range of possible scenarios in terms of (i) project domain, (ii) number of issues, and (iii) developer experience. As such, these projects limit the threats to external validity of our findings.

The projects we use for the analysis are: Aptana Studio (APSTUD)[1], a web development IDE; Dnn Platform (DNN)[2], a web content management system; Apache Mesos (MESOS)[3], a cluster manager; Mule (MULE)[4], a lightweight Java-based enterprise service bus and integration platform; Sonatype's Nexus (NEXUS)[5], a repository manager for software "artifacts" required for development; Titanium SDK/ CLI (TIMOB)[6], an SDK and a Node.js-based command-line tool for managing, building, and deploying Titanium projects; Appcelerator Studio (TISTUD)[7], an Integrated Development Environment (IDE); Spring XD (XD)[8], a unified, distributed, and extensible system for data ingestion, real time analytics, batch processing, and data export.

The number of selected issues ranges from 228 (APSTUD) to 1215 (TISTUD), amounting to 5607 issues in total.

The projects involve developers with a different level of experience in story point estimation. Figure 2 shows, for each project, how many developers have been working with issues that required an estimation. As we can see, there are a lot of seemingly unexperienced estimators, i.e. developers that have been working only on 1-5 issues with story points.

3.2 Issue Selection

To train and test the classifier, issues which satisfy the following conditions are automatically selected:

1. Story points must have been assigned once and never updated afterward. In fact, if the story points estimate gets updated, it may mean that the initial version of the issue report had misleading information, which would confuse the classifier. This explains why we filter out such issue reports.

[1]http://www.aptana.com/

[2]http://www.dnnsoftware.com/products

[3]http://mesos.apache.org/

[4]https://www.mulesoft.com/resources/esb/what-mule-esb

[5]http://www.sonatype.org/nexus/

[6]https://jira.appcelerator.org/browse/TIMOB

[7]http://goo.gl/vGu8k1

[8]http://projects.spring.io/spring-xd/

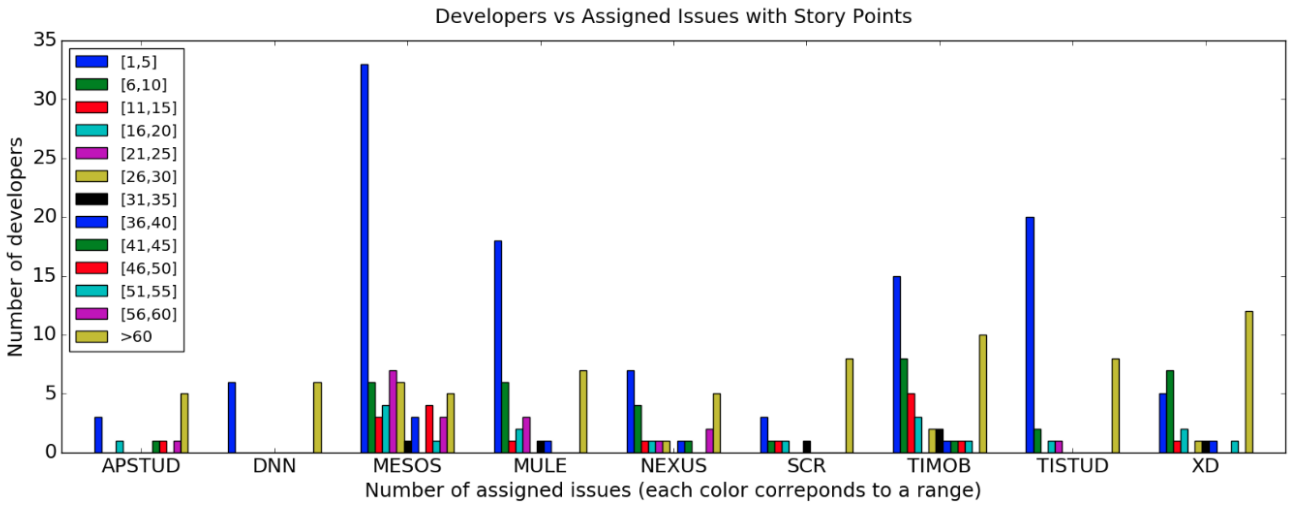

Figure 2: Number of developers vs the total number of issues assigned to them across the projects under study. Only estimated issues (i.e., issues with story points) are considered.

2. The issue must be *addressed*. We consider an issue addressed when its *Status* is set to `Closed` (or similar, e.g. `Fixed`, `Completed`) and its resolution field is set to `Fixed` (or similar, e.g. `Done`, `Completed`). Note that fields such as *Story Points* and *Description* may be adjusted or updated at any given time. However, once the issue is addressed updates rarely happen. For instance, in the industrial project this event happens for less than 4% (49/1368) of the issues. Here as for the other projects, we filter out issue reports not addressed, because they are likely to be unstable, hence they might confuse the classifier.
3. Once the story points are assigned, the *informative* fields of the issue (i) must be already set and (ii) their value must not have been changed afterward. We define informative fields: *Issue Type*, *Description*, *Summary*, and *Component/s*. We filter out issues whose informative fields are updated after story points initialization because they, again, are likely to represent unstable issues.
4. The values in the story points field must correspond to one of those included in the Planning Poker cards set —0, 0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞. We consider values out of the series as erroneous entries and filter them out of the dataset.

Prior to issue filtering, the dataset counts 16,523 issues. After applying the first three filtering criteria, the issue number drops to 5,730. Finally, after filtering out issues not in the Planning Poker card set, 5607 issues are left. We only provide these intermediate values since (i) the first three conditions can be applied in any order, and (ii) different filters may filter out overlapping sets of issues. The total number of issues for each project and type is shown in Table 1.

3.3 Feature Extraction

Certain pre-processing steps are necessary before using a machine learning classifier. In this section, we describe how features were automatically extracted from the issue report fields along with the type of performed pre-processing.

Summary and Description

We use the fields *Summary* and *Description* since (i) they are used by developers for effort estimation and (ii) they have been shown to be effective for story point estimation by Abramhamsson et al. [3].

The textual information from summary and description are joined together to obtain a single text that we will refer to, from now on, as the *context*. Developers can include code snippets in the issue description; as a consequence, the context has always one part hosting natural language text, and sometimes another hosting code. The code snippets are analyzed separately from the rest of the context, since coding words may have a different meaning from those found in natural language text. For instance, "Kill bill" in natural language has a completely different meaning than the statement "bill.kill()" for stopping a process.

To extract features from the context, we employ what is known as a *term frequency – inverse document frequency transformation* (abbreviated as TF-IDF from here on), first on the natural language text and then on the code portion. As TF-IDF needs to be applied, the mono-grams and bi-grams vocabularies corresponding to each of the two text corpus (natural language text and code snippets) are separately built. Stop-words are then filtered out. Stemming is not applied since it revealed to be detrimental to the quality of the estimation.

After building the n-grams vocabulary, term occurrences are computed to transform the contexts with TF-IDF. Namely, a numerical feature is associated to the occurrences of each term in a context. Since we use binary detection, each feature is initially set to either 1 (if found in the context) or 0 (if not found). Afterward, each feature is weighted by dividing it by the n-gram frequency across the corpus.

At this stage, we have a matrix where each row corresponds to an issue, and each element of the row corresponds to the numerical feature associated to a specific term in the issue context. As we obtain one matrix from applying TF-IDF to the natural language portion and another from the code portion, the two matrix are joined to obtain a single one, which we refer to as the feature matrix (rows are issues,

columns are features). The feature matrix is subsequently extended with the features extracted from the *Component/s* and the *Issue Type* fields, and the document size.

Component/s and Issue Type

An issue related to a specific component can have, among its JIRA fields, a tag containing related component(s) name(s). After interviewing the developers of the industrial project, we learned that the fields *Component/s* and *Issue Type* were prime candidates for the story point estimation. This was confirmed by our tests, that eventually lead us to exploit the two fields to extract features. We assigned a numerical feature to each component and issue type, corresponding to 1 or 0 if it is used or not in the issue report, respectively.

3.4 Feature Selection

To reduce the computational time required for estimation we use two approaches. First a univariate feature selection on the feature matrix is performed, which selects features on the basis of their statistical distribution. Then, a selection based on the classifier is applied. The latter uses the coefficients assigned to the features by the classifier to rank them according to their importance.

4. EVALUATION

Once the issues are extracted from the issue tracking system and transformed in a suitable format for training, we evaluate the classifier performance.

4.1 Classifier Validation

To validate our classifier, we used a 10-fold cross-validation since (i) cross-validation reduces overfitting, and (ii) using 10 folds requires less computational time than a higher number of folds, with negligible differences in the results.

4.2 Performance Metrics

To evaluate the performance of the classifier (in terms of estimates) we compute the Mean Magnitude of Relative Error (MMRE) and the accuracy. The former is frequently adopted for evaluating cost estimations in agile software projects [30]. The latter is computed to simplify comparisons with the related works.

The MMRE is defined as the mean value of the Magnitude of Relative Error (MRE) across all the estimates. The MRE definition is:

$$MRE = \frac{|y - \hat{y}|}{y} \quad (1)$$

In our study, y are the story points recorded in the dataset and $\hat{y}$ is the estimate computed by our model.

Accuracy is simply defined as the ratio between the number of correct estimates and the total number of estimates. Total and correct estimates are also reported in the results.

In addition, the confusion matrix was also used to get a more detailed insight into the industrial project [18]. The confusion matrix reports the correct estimates along the main diagonal, whereas the other elements along each row pinpoint which class the incorrect estimates were confused with.

5. RESULTS AND DISCUSSION

This section presents the results of the experiment on the industrial and the open source projects. For each research question we first discuss its motivation, followed by the approach used and, finally, the results.

Table 3: Performance metrics for the industrial project (699 issues) with SVM, NB, KNN, and DT estimators

Estimator	Correct estimates	Accuracy	MMRE
SVM	413	0.59	0.50
NB	309	0.44	0.85
KNN	249	0.36	0.70
DT	158	0.23	0.98

RQ1. What performance can be expected from a machine learning classifier when estimating story points?

Motivation. Since we aim at providing real-time support to the development team in story point estimation, we need to identify a well-suited classifier for the task, also taking into account the estimation time. Hence, we need to evaluate the classifier performance in terms of both correctness of the estimation (e.g. MMRE) and computational time.

Approach. We investigate which typical machine learning classifier is the best one for estimating story points. For this purpose we compare the following algorithms: Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Decision Tree (DT), and Naive Bayes (NB). The algorithms SVM, KNN, and DT were chosen since they are generally well suited for text classification applications. Moreover, the KNN and DT produce easily interpretable classifiers. Finally, the algorithm NB was chosen due to its simplicity and yet high accuracy [27].

Once identified the best algorithm, we use it on the industrial and open source projects. It is beyond the scope of this paper to present all of the analysis results; rather, we present the results achieved with the best algorithm. The other results are available in the replication package[9].

To evaluate the performance of the classifier we compute both MMRE and the accuracy to simplify comparisons with related works. In addition, we compare our results with a ZeroR classifier, a classifier that always selects as the outcome the most represented class. We use the ZeroR to verify how much better is our classifier compared to arbitrary approximation. We complete the discussion by providing the computational time required by the estimation.

Findings. We present the results achieved on the industrial project and on the open source projects in two separate sections.

Industrial Project. Table 3 reports the performance obtained on the industrial project using different algorithms. The SVM achieves the best performance having: MMRE 0.5, 413 correct estimates, and 59% accuracy. This result is aligned with the state of the art since the SVM has been proven to be well suited for text categorization [20].

To better understand the results achieved by the SVM classifier, we report in Table 4 the confusion matrix for the industrial project. Here, we can observe that, for any given class, most issues are correctly classified (main diagonal). In addition, when the classifier misclassified an issue, estimated story points fell in adjacent classes. However, classes 8 and 13 do not follow this positive trend, most likely because there were not enough training data; these categories only

[9]http://goo.gl/mLe26i

Table 4: Confusion Matrix for the industrial project (SCR)

	<1	**1**	**2**	**3**	**5**	**8**	**13**
<1	**46**	13	16	11	3	0	0
1	11	**116**	26	11	10	0	0
2	12	27	**119**	24	4	1	0
3	7	16	25	**100**	6	3	0
5	3	11	8	16	**30**	4	0
8	1	5	5	2	3	**2**	0
13	0	0	1	1	0	0	**0**

Table 5: SVM performance metrics, with total and correct estimates.

Project	**Total est.**	**Correct est.**	**Accuracy**	**MMRE**
APSTUD	228	143	0.63	0.61
XD	360	223	0.62	0.42
MESOS	387	223	0.58	0.39
NEXUS	421	341	0.81	0.16
TIMOB	634	380	0.6	0.6
SCR	**699**	**413**	**0.59**	**0.5**
MULE	805	416	0.52	1.07
DNN	858	669	0.78	0.16
TISTUD	1215	810	0.67	0.35
mean	*623*	*402*	*0.64*	*0.47*

account for 2.87% of the total amount of issues. The result is encouraging since it shows that estimates tend to converge toward the correct estimation.

Open Source Projects. Table 5 presents the performance achieved by the SVM classifier on the eight open source projects. For comparison, the results achieved on the industrial project are reported in bold. Here, we can notice that the worst performance of the classifier is a MMRE of 1.07 on the MULE project. Leaving out this exception, APSTUD and DNN respectively represent the (second) worst and the best case study for the SVM classifier. For APSTUD, accuracy and MMRE are, respectively, 0.63 and 0.61, whereas for DNN, accuracy and MMRE are, respectively, 0.78 and 0.16. In light of the foregoing, the performance on the industrial project falls in between.

In order to compare our classifier with the state of the art, we can refer to the work by Abrahamsson et al., where a classifier for story point estimation is presented [3]. Inferring the development time via story points, they compute the MMRE as the difference between the actual (provided by developers after implementing the task) and estimated development time (provided by the classifier) [3]. Their classifier, tested on two industrial projects, led to a MMRE higher (worst) than that achieved by our classifier.

Another comparison is made with the work of Augen [7], who ran an experiment on 19 developers using Planning Poker, computing the MMRE by considering the story points assigned by developers before and after task implementation. He achieved an MMRE of 0.48 [7], which is aligned with the average in our analysis (0.47). From this point of view, our classifier performs as good as a human, hence is sufficiently reliable to serve in an actual poker planning session.

Table 6: ZeroR performance metrics, with total and correct estimates.

Project	**Total est.**	**Correct est.**	**Accuracy**	**MMRE**
APSTUD	228	70	0.31	1.2
XD	360	117	0.32	0.45
MESOS	387	120	0.31	0.44
NEXUS	421	199	0.47	0.46
TIMOB	634	192	0.3	0.92
SCR	**699**	**187**	**0.27**	**0.79**
MULE	805	171	0.21	0.62
DNN	858	375	0.44	0.33
TISTUD	1215	489	0.4	0.52
mean	*623*	*213*	*0.34*	*0.64*

We evaluate the reliability of the SVM classifier by comparing its performance with the ZeroR classifier (Table 6). The last row of Table 6 shows that the SVM classifier is better than the ZeroR, since it has, on average: (i) a two times higher (0.64/0.34) accuracy and (ii) a MMRE roughly 25% (0.47/0.64) lower. On the industrial project, the performance is even better: the accuracy is 2.18 higher, and MMRE is around 63% of the MMRE achieved by the ZeroR. The results achieved by the SVM and the ZeroR classifiers can be partially explained looking at the distribution of story points in both projects. MULE has almost the same number of issues in each of the classes <1, 1, 3, 5, and 8; on the contrary, DNN has more than 0.40 of the issues just in class 1, and more than 0.33 in class 2. In view of these differences, it is understandable why both the ZeroR classifier and, to a lesser extent, the SVM classifier scores a better performance on the DNN project.

All of the analysis were performed on a machine equipped with an Intel® Core™ i7-6500U quad-core CPU at 2.50GHz, with 8 GB DDR3 SDRAM. Running a 10-fold cross-validation on TISTUD, the project with more than 1200 issues, requires only 6 seconds. In addition, once the system is set up, the total required time for retrieving a new issue from the issue tracking system repository and estimating its story points ranges from 8 to 15 seconds. Thus, the proposed classifier is fast enough to be integrated in project planning sessions (e.g. Planning Poker).

A SVM classifier for story point estimation can achieve a MMRE spanning between 0.16 and 0.61, namely comparable with developer estimates. Moreover, the estimation process takes less than 6 seconds.

RQ2. Which attributes of an issue report are relevant for estimating story points?

Motivation. We want to analyze the top 10 features used by the classifier. We are interested in contextualizing these features within the project from where they come from in order to show their meaningfulness and validate the consistency of the results obtained in RQ1. For this reason, we only refer to the SVM classifier, since it achieves the best performance.

Table 7: Top 10 features (positions 1-5)

Project	1	2	3	4	5
APSTUD	(typestory, 6.62)	(change, 5.84)	(invoke, 5.81)	(componenthtml, 5.5)	(against, 5.43)
DNN	(typestory, 10.1)	(typebug, 9.38)	(dnn search, 6.6)	(for user, 6.56)	(from page, 6.53)
MESOS	(what, 5.38)	(on, 5.23)	(necessary, 5.17)	(statistics, 5.14)	(in, 5.1)
MULE	(execution, 10.63)	(componentextensionsapi, 10.49)	(async, 10.3)	(consistent, 10.17)	(consistency, 9.95)
NEXUS	(minor, 5.24)	(repository metadata, 4.93)	(seeing, 4.85)	(detected, 4.83)	(clicking, 4.83)
SCR	**(tests for, 7.8)**	**(scenario, 7.48)**	**(services, 7.33)**	**(included, 7.29)**	**(started, 7.25)**
TIMOB	(txt, 6.72)	(classic, 6.41)	(create an, 6.28)	(we have, 6.02)	(great, 5.73)
TISTUD	(npe, 9.11)	(each, 7.7)	(update process, 7.56)	(switching, 7.31)	(typebug, 7.23)
XD	(module, 8.11)	(consider, 7.77)	(changed, 7.54)	(containers, 7.5)	(typebug, 7.47)

Table 8: Top 10 features (positions 6-10)

Project	6	7	8	9	10
APSTUD	(need to, 5.37)	(down, 5.34)	(at, 5.28)	(now, 5.26)	(top, 5.25)
DNN	(get, 6.4)	(usage, 6.39)	(platform build, 6.37)	(executed, 6.33)	(description, 6.31)
MESOS	(fetchercachetest, 5.05)	(make, 4.93)	(verify, 4.9)	(attempts, 4.9)	(run, 4.81)
MULE	(java, 9.83)	(concurrent, 9.69)	(to read, 9.67)	(including, 9.59)	(payload value, 9.56)
NEXUS	(properly, 4.72)	(in case, 4.59)	(hide, 4.55)	(not, 4.55)	(testsuite, 4.46)
SCR	**(cache, 7.1)**	**(vm, 7.06)**	**(fine, 6.97)**	**(do not, 6.87)**	**(next to, 6.77)**
TIMOB	(code, 5.7)	(get, 5.67)	(should use, 5.55)	(machine, 5.54)	(root, 5.52)
TISTUD	(to remove, 7.05)	(detection, 6.98)	(defined, 6.98)	(case where, 6.78)	(read, 6.77)
XD	(because, 7.2)	(so we, 7.18)	(to get, 7.11)	(supports, 7.09)	(out, 7.07)

Approach. We sort the features used by the classifier in RQ1 by their weight, in descending order. With regard to the weight assigned for feature ranking, the absolute sum of the features coefficients over all the classes was computed as a proxy of the feature importance. Finally, we provide a qualitative analysis of the keywords used across the projects.
Findings. Tables 7 and 8 report the top 10 features for each project in the dataset. Each feature is reported along with its weight. The Tables present text in the form of single keywords (e.g. `UTC`), bigrams (e.g. `ci bug`), components (e.g. `component html`), and issue type (e.g. `type bug`). Being in the top 10, all the previous informative fields are extremely relevant for the classification. To a lesser extent, also the length of the description is relevant for classification purposes. With TIMOB, the feature length is ranked 322, whereas with MESOS its rank is 472. We confirm the results found by Abhramsson et al. [3], namely, that textual information and its length are effective predictors for story point estimation.

For each project, a different set of features appears in the top 10. This result is consistent with the fact that projects have different goals and a specific vocabulary. For instance, the DNN project is a content management system. Here views and user searches are core aspects, and keywords such as `from page` and `dnn search` reflect this relevance. Similarly the MULE project, a runtime engine for combining data and application integration across legacy systems, uses keywords such as `consistency, async, concurrent`. Finally, we can also notice that there are common features across projects, such as the maintenance type (`type bug`, `type story`). This result is consistent with Murgia et. al, who found that the issue fixing time is affected by the type of maintenance performed [22]. Except for the maintenance type, projects tend to have specific keywords. As a consequence, a machine for story point estimation requires a specific training on the project issues.

> *In an issue report, the fields containing summary, description, names of related components, and issue type provide relevant features for story point estimation. Most frequently, these features are project dependent.*

RQ3. How much training data is needed to obtain a satisfactory estimation?

Motivation. We are interested in introducing the classifier during the developers' estimation process. However, in the first stage of the project development, the number of issues per class may not be enough to ensure proper training. As a result, the provided estimates would be poor and unsatisfactory. Therefore, we investigate the amount of training data needed to obtain a sufficiently precise classifier.
Approach. We use the SVM classifier since it achieves the best performance in RQ1. For this classifier, we investigate how the size of the training set affects its "learning curve". The learning curve describes how the performance of the classifier (y-axis) evolves along with the number of issues considered (x-axis). To reproduce a real scenario, we first sort all issues by their estimation age, namely by the time of story points initialization. Then we divide the issues in blocks[10] of 20. Using the first block, we evaluate the MMRE by means of a 10-fold cross-validation. We repeat this procedure progressively considering 40, 60, and so on issues. By using the 10-fold cross-validation, we can evaluate the evolution of the classifier performance against the number of

[10]We consider a block size of 20 as a trade-off. In fact, we observed that smaller blocks (e.g. 10 issues) generate too many points and thus a clotted graph. On the other hand, for small projects, bigger blocks (e.g. 50 issues) generate too few points for any trend to be observable.

blocks considered, and compare it with the results reported in RQ1.

Findings. Figure 3 shows the evolution of the MMRE for all 9 projects. Here we can observe that below 200 issues, many projects have an erratic evolution of the MMRE. For instance, the projects SCR, MESOS, APSTUD, TIMOB show MMRE variations higher than 0.2. Over 300 issues, with the exception of MULE, all projects achieve a MMRE lower than 0.62. In particular, the projects DNN and NEXUS reach a MMRE lower than 0.2.

In any project, estimates quality changes along with the number of issues used for training. Except for TISTUD, with more than 300 issues we cannot observe a clear decreasing (nor increasing) trend of the MMRE. Yet, we can only speculate that the value of MMRE would stay "limited". Indeed, with SCR, TISTUD, TIMOB, NEXUS, DNN, XD, and MESOS, the MMRE shows a variation[11] which is less than or equal to 0.135. It is then pivotal to accompany the classifier evaluation with the number of issues used for its training.

In the graph, we may observe several peaks. We analyzed a few cases and, as a representative one, we took the peak value of TISTUD when the number of issues is 280. We found out that such variations are mainly caused by misclassifications involving the inclusion of issues with a higher number of story points[12].

It is better to train the classifier with more than 200 issues before employing it for story point estimation. If 0.61 is a satisfactory value for the MMRE, the team can generally start using the classifier with just 300 issues. However, the performance achieved depends on the project.

6. THREATS TO VALIDITY

In this section we discuss factors which can hinder the validity of our study, and the measures taken to counteract them.

Construct validity. MMRE is not necessarily the most suitable metric to evaluate a story point estimator. We chose the MMRE since its wide adoption allows for an easy comparison with the state of the art. Nevertheless, as story point classes are generally unbalanced, and misclassifications are supposed to have a different weight according to the true and predicted classes, the use of a specifically tailored metric for story point estimation would have been advisable.

Internal validity. The experiments were performed under the assumption of a causal relationship between the textual information on an issue report and the story point estimation. However, there are confounding factors which can influence the estimation process, such as company political pressures, the organization estimation process, or the expertise of the issue reporter. Developer's expertise, for instance, can lead to poorly written issue reports. Nevertheless, as pointed out in the experimental setup, we applied a whole range of selection criteria specifically aimed at minimizing the effects of misleading issue reports.

External validity. Given the limited number of projects in our study, namely 8 open source and 1 industrial, we can not assume that the observed estimation performance holds true for other projects. More specifically, estimation processes and company politics might lead to different results. However, since we selected heterogeneous software projects with highly varied story point distributions, we are confident that these threats to external validity are limited.

In addition, we have to remark that we only relied on JIRA repositories. However, the proposed estimator is based on textual information which can be extracted from any issue tracking system, hence basically requiring to only adapt the software to a different API.

Another limitation might be the lack of issue reports at project start. Under this condition, the estimator can be set to only provide estimates when its confidence is higher than a specified threshold.

7. RELATED WORK

A realistic and reliable effort estimation process is crucial for a correct project planning. In the agile domain, researchers found that effort is commonly estimated in story points, and that the estimation techniques based on human judgment, such as the Planning Poker, are the most frequently applied [30]. Several studies discussed the problems associated to these estimation strategies.

Hughes showed that judgmental bias of either social or cognitive nature can affect estimations. Social judgmental bias, which may for instance be caused by managerial pressure, can lead estimators to reduce the information used with the aim of simplifying a development task to one of manageable proportions [15]. Abdel-Hamid et al. focused on the effects of schedule compression and pointed out that schedule pressure via underestimation turns into a higher development cost and bug rate [1]. Finally, DeMarco reported that human estimators may tend to underestimate the time required to do a task when they are appointed as assignees for that task [12].

Previous results give room to the adoption, during developer estimation meetings, of a machine-mediated estimation capable of grounding its outcome to objective data. Regarding this matter, several studies have investigated the adoption of a machine for effort estimation. Generally, these approaches involve machine learning models which, via different algorithms, estimate the time required to fix a bug [2][21][31]. However, there are also cases where Neural Networks [33] and association rules have been adopted [28]. Despite the interest on issue effort estimation, none of these studies estimate the effort in terms of story points. As far as the authors know, only Abrahamsson et al. built a classifier for estimating story points [3]. They used the classifier on the user stories provided by an agile company with more than 1325 issues[13] [3]. The best result they achieved

[11]Using TISTUD, the project with the highest variation of MMRE after 300 issues, as a reference, we observe that 0.6 and 0.33 are the highest and lowest MMRE values, namely, at a distance of 0.27 (or ± 0.135) MMRE with respect to the value between 0.6 and 0.33.

[12]This result was somehow expected since the classifier cannot correctly estimate story points for class of issue reports barely represented. Even a slight change in their population at the beginning of the learning curve may lead to a considerable change in MMRE.

[13]In the paper Abrahamsson et al. use the classifier also on a commercial project. However, this project has only 13 issues.

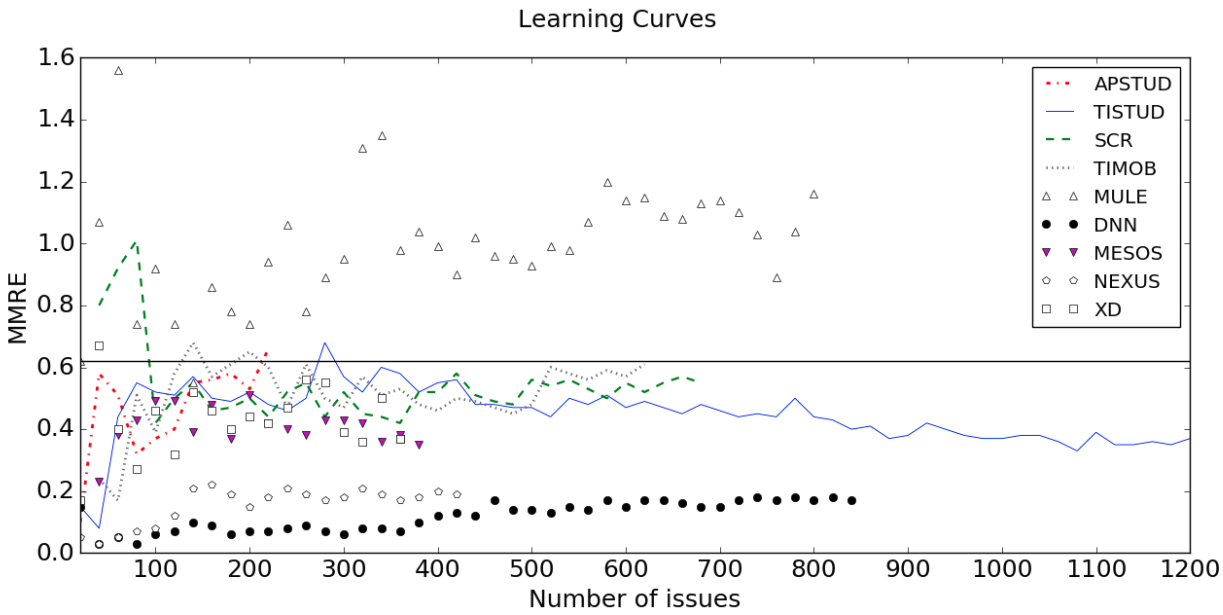

Figure 3: Learning curves with line at MMRE=0.62

was a MMRE of 0.9, obtained by using the SVM algorithm. Similarly to Abrahamsson et al., we tried different machine learning classifiers to estimate story points. We confirm that the SVM algorithm is the best one for processing the issue reports; however, through leveraging different features, we achieved better results. Using more than 300 issues, we obtained a MMRE lower than 0.61 in 8 out of 9 analyzed projects. In addition, we also discussed how the number of issues affects the MMRE.

On the human side, Augen [7] analyzed developer-mediated estimation of user stories. He focused on 101 estimates reported by an XP development team during release planning. He compared the story points estimated by developers at the initial release planning meeting with the story points effectively assigned once the task was implemented. The study reports that developer estimations achieve a MMRE of 0.48. From this point of view, our classifier —reaching (on average) a MMRE of 0.46— produces estimates aligned with those provided by developers.

8. CONCLUSION AND FUTURE WORK

Effort estimation is a cornerstone of every software project. In the agile context, effort estimation is frequently done via *story points* (an estimate which is a proxy for the effort needed for the completion of a task) and group discussions (to achieve consensus on a given estimate). All these estimation techniques are based on human judgment, that has clear advantages but some drawbacks as well. To counteract these drawbacks, we explore the feasibility of a tool for supporting human judgment during story point estimation. More specifically, we propose a machine learning classifier capable of processing an issue report and providing an effort estimate in terms of story points in real-time. Based on an analysis involving one industrial project and eight open source projects, we demonstrate that such tool support is feasible. Indeed, we implement a classifier which achieves —for all projects but one— an MMRE spanning from 0.16 to 0.61, after an initial training of at least 300 issues. Moreover, once the system is set up, the time required for estimating a new issue ranges from 8 to 15 seconds.

We also found that issue type, summary, description, and related components carry project dependent information that contain relevant features for story point estimation. Finally, an MMRE of 0.61 is generally obtainable by training the estimator on more than 300 issues.

As a future development, we plan to improve the estimator usability. For this reason, the estimator will (i) be implemented as a JIRA Cloud add-on [19], and (ii) be able to highlight, in the issue report, the most important keywords adopted for the estimation. The latter functionality would reduce the chances of having relevant keywords passing unnoticed. Further down the line, we plan to investigate (i) the role of emotions in the estimation process [24] and (ii) developer-estimator interaction. For the latter, it is indeed known from literature that humans may trust more their own peers than a machine [23].

9. ACKNOWLEDGMENTS

Simone Porru gratefully acknowledges Sardinia Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 Axis IV Human Resources, Objective l.3, Line of Activity l.3.1). The Second sponsoring organisation is the Institute for the Promotion of Innovation through Science and Technology in Flanders by means of a project entitled Change-centric Quality Assurance (CHAQ) with number 120028. We also thank Tars Joris, scrum master and team leader with more than 10 years of experience in industry, for the feedback provided during the research.

10. REFERENCES

[1] T. K. Abdel-Hamid. Investigating the cost/schedule trade-off in software development. *Software, IEEE*,

7(1):97–105, 1990.

[2] W. AbdelMoez, M. Kholief, and F. M. Elsalmy. Improving bug fix-time prediction model by filtering out outliers. In *Proceedings of the 2013 International Conference on Technological Advances in Electrical, Electronics and Computer Engineering*, pages 359–364, May 2013.

[3] P. Abrahamsson, I. Fronza, R. Moser, J. Vlasenko, and W. Pedrycz. Predicting development effort from user stories. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, pages 400–403. IEEE, 2011.

[4] J. Aranda and S. Easterbrook. Anchoring and adjustment in software estimation. In *Proceedings of the 2005 European Software Engineering Conference Held Jointly with the 2005 International Symposium on Foundations of Software Engineering*, pages 346–355. ACM, 2005.

[5] E. Aronson, T. D. Wilson, and R. M. Akert. *Social Psychology (3rd Edition)*. Pearson, 1999.

[6] R. L. Atkinson, R. C. Atkinson, E. E. Smith, D. J. Bem, and S. Nolen-Hoeksema. *Hilgard's Introduction to Psychology (12th Edition)*. Orlando: Harcourt Brace College Publishers, 1996.

[7] N. C. Augen. An empirical study of using planning poker for user story estimation. In *Agile Conference, 2006*, pages 9–pp. IEEE, 2006.

[8] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development, 2001.

[9] L. C. Briand. On the many ways software engineering can benefit from knowledge engineering. In *Proceedings of the 2002 international conference on Software engineering and knowledge engineering*, pages 3–6. ACM, 2002.

[10] R. Brown. *Group Processes: Dynamics Within and Between Groups*. Wiley-Blackwell, 2000.

[11] M. Cohn. It's effort, not complexity. https://goo.gl/nNYlL1, 2010. Accessed: 2016-06-02.

[12] T. DeMarco. *Controlling software projects: Management, measurement, and estimates*. Prentice Hall PTR, 1986.

[13] Why do the cards deviate slightly from the fibonacci sequence? http://goo.gl/xgs7PF. Accessed: 2016-07-13.

[14] J. Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, 3, 2002.

[15] R. T. Hughes. Expert judgement as an estimating method. *Information and Software Technology*, 38(2):67–75, 1996.

[16] Github issues tutorial. https://goo.gl/OR1gwr. Accessed: 2016-07-13.

[17] What is an issue - atlassian documentation. https://goo.gl/JduWpL. Accessed: 2016-07-13.

[18] N. Japkowicz and M. Shah. *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[19] Jira - pros and cons of cloud vs. server. https://goo.gl/5KF2rI. Accessed: 2016-07-13.

[20] T. Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[21] L. Marks, Y. Zou, and A. E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 2011 International Conference on Predictive Models in Software Engineering*, page 11. ACM, 2011.

[22] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi. On the influence of maintenance activity types on the issue resolution time. In *Proceedings of the 2014 International Conference on Predictive Models in Software Engineering*, pages 12–21. ACM, 2014.

[23] A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu. Among the machines: Human-bot interaction on social Q&A websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 1272–1279, New York, NY, USA, 2016. ACM.

[24] A. Murgia, P. Tourani, B. Adams, and M. Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 262–271, New York, NY, USA, 2014. ACM.

[25] J. W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4):246–256, 2004.

[26] Planning poker 3.0. https://goo.gl/Tl5mws. Accessed: 2016-06-16.

[27] I. Rish. An empirical study of the naive bayes classifier. Technical report, 2001.

[28] Q. Song, M. Shepperd, M. Cartwright, and C. Mair. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, 32(2):69–82, 2006.

[29] A. Trendowicz, J. Münch, and R. Jeffery. State of the practice in software effort estimation: a survey and literature review. In *Software Engineering Techniques*, pages 232–245. Springer, 2008.

[30] M. Usman, E. Mendes, F. Weidt, and R. Britto. Effort estimation in agile software development: A systematic literature review. In *Proceedings of the 2014 International Conference on Predictive Models in Software Engineering*, pages 82–91. ACM, 2014.

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

[32] What is a story point? https://goo.gl/Vfb9v1, 2007. Accessed: 2016-06-02.

[33] H. Zeng and D. Rine. Estimation of software defects fix effort using neural networks. In *Proceedings of the 2004 Annual International Conference on Computer Software and Applications*, volume 2, pages 20–21. IEEE, 2004.