Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Descriptions of issues and comments for predicting issue success in software projects

Sandra L. Ramírez-Mora[a,*], Hanna Oktaba[b], Helena Gómez-Adorno[c]

[a] Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Mexico City, Mexico
[b] Facultad de Ciencias, Universidad Nacional Autónoma de México, Mexico City, Mexico
[c] Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Mexico City, Mexico

ARTICLE INFO

Article history:
Received 1 October 2019
Received in revised form 18 February 2020
Accepted 22 May 2020
Available online 27 May 2020

Keywords:
Software development
Software project
Issue success
Machine learning
Issue Tracking System

ABSTRACT

Software development tasks must be performed successfully to achieve software quality and customer satisfaction. Knowing whether software tasks are likely to fail is essential to ensure the success of software projects. Issue Tracking Systems store information of software tasks (issues) and comments, which can be useful to predict issue success; however; almost no research on this topic exists. This work studies the usefulness of textual descriptions of issues and comments for predicting whether issues will be resolved successfully or not. Issues and comments of 588 software projects were extracted from four popular Issue Tracking Systems. Seven machine learning classifiers were trained on 30k issues and more than 120k comments, and more than 6000 experiments were performed to predict the success of three types of issues: bugs, improvements and new features. The results provided evidence that descriptions of issues and comments are useful for predicting issue success with more than 85% of accuracy and precision, and that the predictions of issue success vary over time. Words related to software development were particularly relevant for predicting issue success. Other communication aspects and their relationship to the success of software projects must be researched in detail using data from software tools.

1. Introduction

Software activities, such as the development of new functionalities, the improvement of software, and the correction of defects must be continuously and successfully performed to ensure software quality and to satisfy the needs of customers and users of software products. Predicting whether software activities will be performed successfully is crucial in software projects, especially when a lot of people use software products that constantly evolve. Knowing whether an activity is likely to fail can increase software quality and customer satisfaction in software projects because resources can be managed to ensure the completion of software activities, such as the correction of blocking software defects that are crucial for many stakeholders of software products. Early prediction of task success can also reduce development time because people can make opportune decisions without having to wait until the completion of an activity.

Research on predicting outcomes of software projects often focus on predicting the time or effort to perform software tasks; however, knowing the approximate time in which a software task will be completed may be useless if the task will be unsuccessfully completed and actions for ensuring the success of the task are not performed, so the prediction of issue success is crucial in software engineering.

Software development tools facilitate collaboration, are useful for managing software activities, and store information that can be useful to predict the success or failure of software tasks. Particularly, Jira is a popular software development tool in which software tasks (called "issues") can be recorded and managed. Comments of issues can also be recorded in Jira to provide additional detail about software tasks and collaborate with team members. Jira is often used as an Issue Tracking System (ITS) and many organizations such as Apache and Spring have a public Jira ITS. Comments of issues that are recorded in Jira ITSs usually contain implicit and explicit information about the progress and development of software tasks (issues), so they can be used to detect whether an issue will be successfully resolved or not.

Software teams frequently use web tools (such as ITSs) to communicate; therefore, textual descriptions of comments and issues from Jira ITSs represent an important part of the information that is communicated among customers, users, and developers

* Correspondence to: Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Av. Universidad 3000, Coyoacán, C.P. 04510, Ciudad Universitaria, Mexico City, Mexico.
E-mail addresses: sandra.ramirez@ciencias.unam.mx (S.L. Ramírez-Mora), hanna.oktaba@ciencias.unam.mx (H. Oktaba), helena.gomez@iimas.unam.mx (H. Gómez-Adorno).

of software products. Communication is closely related to the success of software projects, and some works have found a positive impact of quality communication on productivity and project success (Melo et al., 2013; Destefanis et al., 2016; Ramírez-Mora and Oktaba, 2017). Based on the relationship between communication and the success of software projects, and based on the fact that comments reflect the nature of communication in software development and contain implicit and explicit information about the progress of issue resolution, the following hypothesis was formulated: descriptions of issues and comments from Jira ITSs are useful for predicting issue success. From this hypothesis, the following questions were defined to study the usefulness of descriptions of issues and comments for predicting the success of three types of issues (bugs, improvements and new features).

RQ$_1$: Are textual descriptions of issues and comments from Jira ITSs useful to predict issue success in software projects?

RQ$_{1.1}$: What kind of information is useful to predict issue success?

RQ$_{1.2}$: How does the prediction of issue success vary over time?

RQ$_{1.3}$: How does the prediction of issue success vary with respect to bugs, improvements and new features?

This work contributes on the research regarding the early prediction of issue success using descriptions of comments and issues from Jira ITSs. The remainder of this paper is organized as follows: Section 2 describes the research background; Section 3 describes the related work; Section 4 details the research method; Section 5 describes the results; in Section 6, the results and validity of this work are discussed; and in Section 7, conclusions are described and directions for future work are suggested.

2. Background

Software development tools are very useful for communicating and collaborating during software development processes, especially for people that are geographically distributed. During software lifecycle, developers, users, and customers require to report software defects, provide feedback, request new features or improvements, and monitor the progress of software tasks, and software development tools are very useful for these purposes. Knowing the progress of software tasks is very important for users and customers of software products because they need to know in advance whether their requests are being successfully achieved.

Jira is a software development tool widely used by agile teams to plan, track, and release software. In Jira, “issues” are the elemental components of a software project and represent software bugs, project tasks, requirements, improvements or another issue type. Comments of issues can also be recorded in Jira to provide additional information of issues and facilitate the collaboration among people (developers, users, customers) involved in software development activities. Jira is commonly used as an Issue Tracking System (ITS) and many organizations (including Apache, Spring, Hibernate, Atlassian, Red Hat and Fedora) have a public Jira ITS to record and manage issues of their software projects, particularly of their open source projects.

Progress of tasks can be monitored in Jira because issues are continually tagged according to their status during their lifecycle. An issue is initially “open” (created), and when people start working on it, the issue is tagged as “in progress”. When an issue is attended, it is tagged as “resolved”, and if the resolution is accepted, the issue is tagged as “closed”, otherwise, the issue is “reopened”. Authorized stakeholders (developers, project managers, business people, etc.) in a software project can open, resolve, and close issues, and they can indicate issue labels, such as the status and resolution of issues. Fig. 1 shows an example of some possible statuses of an issue during its lifecycle.

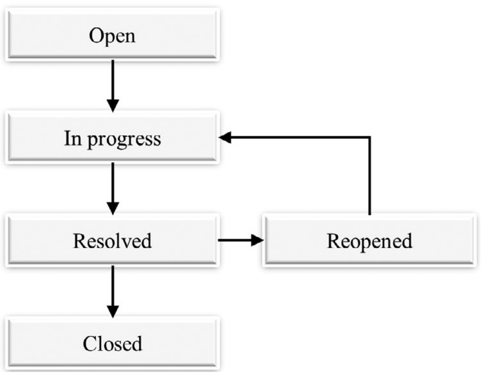


Fig. 1. Example of some possible statuses of an issue during its lifecycle.

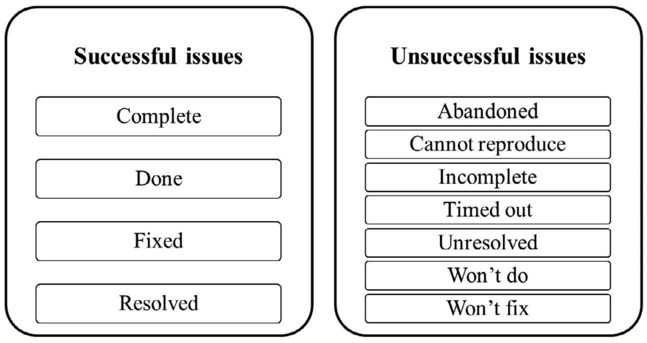


Fig. 2. Usual resolution tags of successful and unsuccessful issues.

When an issue is resolved, it is tagged according to its resolution type (Fig. 2). Issues that have been resolved successfully are usually tagged as “complete”, “done”, “fixed” or “resolved” in Jira ITSs; however, there are many issues that are not resolved successfully, and they are typically tagged with one of the following labels.

- “Abandoned”. Indicates that an issue was abandoned.
- “Cannot reproduce”. Indicates that an issue cannot be reproduced.
- “Incomplete”. Indicates that an issue is not described completely.
- “Timed out”. Indicates that an issue was closed due to lack of response.
- “Unresolved”. Indicates that an issue was not resolved.
- “Won’t do”. Indicates that an issue won’t be actioned.
- “Won’t fix”. Indicates that the problem described is an issue which will never be fixed.

In Jira ITSs, each issue and comment is stored with a description, a reporter, a unique identifier and the date of report. Descriptions of comments and issues are texts that have lexical, syntactic and semantic characteristics, so they can be studied using Natural Language Processing (NLP) techniques. Descriptions of comments and issues often include technical information (URLs, fragments of code, software specifications, file directories), information about the progress of software tasks, and many of them include information regarding interactions and collaboration among people.

Comments and issues from Jira ITSs represent part of the communication that is performed during software development activities. Some communication models represent communication as an action in which a sender (source or speaker) transmits a message through some channel to a receiver (Littlejohn and Foss, 2009). Considering this kind of models, the reporter of a comment or issue is the sender, the description of the comment or issue is the message, and the people who read the comment or issue are the receivers in a communication process. Comments and issues are usually reported and read by developers, project managers, customers and users of software products.

Communication is a key factor for the success of software projects and the productivity improvement, and some previous works have provided evidence of this. In a systematic mapping study (Ramírez-Mora and Oktaba, 2017), communication was found to be a factor that impacts productivity in agile software development. Fagerholm et al. (2015) concluded that enhancing performance experiences of software teams requires integration of soft factors, such as communication. Hoegl and Gemuenden (2001) and Lindsjørn et al. (2016) found a strong correlation between teamwork quality (which includes communication) and the success of software development teams. Nasir and Sahibuddin (2011) conducted a comparative study and found that communication is one of the critical success factors for software projects. The results of Destefanis et al. (2016) showed that the level of politeness in the communication process among developers does have an effect on the time required to fix issues in software projects. McLeod and MacDonell (2011) conducted a survey of research regarding the factors that affect software systems development project outcomes, and found that communication is often perceived as an important dimension of the interaction between users and development staff, essential for effective functioning of the project team, and a key factor in system success; these declarations are based on the works of Akkermans and van Helden (2002), Butler (2003), Butler and Fitzgerald (2001), Hartwick and Barki (2001), Sawyer and Guinan (1998), and Somers and Nelson (2001). Garousi et al. (2019) studied the correlation of critical success factors with the success of software projects and found that the higher the quality of internal team communication, the higher the team building and team dynamics at the end of a project.

Based on the evidence of the close relationship between communication and the success of software projects, and the kind of information that is usually included in comments and issues (such as the progress of software tasks) from ITSs, the prediction of issue success could be performed using the textual descriptions of issues and comments, which are communicated during software development activities. The prediction of issue success is a concern of stakeholders (developers, customers, users) of software products, who frequently need to know whether an issue will be successfully addressed or not; however, scarce research on this topic exists.

3. Related work

Researchers and practitioners are often interested on predicting and studying outcomes of software processes and tasks (such as time, effort, cost and success) and characteristics of software products (such as quality, faults and bugs). Murgia et al. (2014) investigated the influence of the maintenance activities types on issue resolution time using data from projects in GitHub.

Machine learning techniques are usually used to predict software development aspects. Catal (2011) conducted a literature review of the trends on software fault prediction and found that supervised algorithms of machine learning and software metrics can be used to build prediction models. Hall et al. (2012) conducted a systematic literature review to investigate how the context, variables and modeling techniques influence the performance of fault prediction models. They found that many different variables have been used (including metrics of process and products, metrics relating to developers, and texts of source code) and concluded that more studies with a reliable methodology and detailed content are needed.

Most of the works that have reported predictions in software development have used data of software artifacts and processes, such as the work of Guo et al. (2010). Suma et al. (2014) used a machine learning classifier to predict software defects using data of software defects, software size and project development time. In the work of Choetkiertikul et al. (2018), an approach was proposed for predicting delivery capacity for software development iterations applying machine learning techniques and using data from previous iterations and issues.

Few works that have used data of human factors for predicting outcomes of software tasks exist. Duc Anh et al. (2011) assessed different types of issue lead time prediction models using human factor measures that were collected from ITSs. These data include the experience of reporters and assignees of issues, the number of comments of issues, and the number of the involved stakeholders in an issue. The results indicated that the number of stakeholders and the average of the lead time of previous issues (resolved by developers) are important variables in constructing prediction models of issue lead time; however, authors concluded that more variables should be explored to achieve better prediction of performance. Ortu et al. (2015) studied the relationship between "affectiveness" (sentiments, emotions and politeness) of developers and the time to fix issues using machine learning techniques. The authors used data of issues and comments and found that happy developers are likely to fix issues in short time and that negative emotions are linked with longer issue fixing time.

Some works that report the use of texts to perform predictions of software tasks outcomes and software products characteristics exist; however, scarce research regarding the prediction of software tasks success exists. Di Sorbo et al. (2019) investigated the nature of "won't fix" issues in GitHub, performed predictions of "won't fix" issues using machine learning techniques and textual features (titles and descriptions of reported issues), and used such textual features to identify the most important words in the issue titles and descriptions. Fronza et al. (2013) described an approach to predict failures of software systems based on log files using text analysis techniques and machine learning algorithms. Binkley et al. (2009) applied three language-processing measures (based on the percentage of natural language words in code, the percentage of identifiers that violate syntactic conciseness and consistency rules, and the similarity between a module's comments and its code) to the problem of fault prediction using data from Mozilla and their results demonstrated the usefulness of these measures for fault prediction. Valdivia-Garcia et al. (2018) studied bugs that block the fixing of other bugs in eight open source projects and proposed a model to predict them using data of bugs such as priority, severity, descriptions and comments. They found that the description and the comments included in the bugs were the most important factors for predicting blocking bugs.

In general, almost no works on predicting issue success exist, and few works have reported the use of human factors and interaction aspects (such as natural language and communication) to perform predictions in software development. Almost no works that have used textual descriptions (and its lexical, syntactic and semantic characteristics) to perform success predictions exist. In addition, scarce research on the identification of relevant information from texts to predict issue success exists, and works that consider issue types and time as variables for the prediction of issue success are lacking. The objective of this work is to contribute to the investigation on these topics.

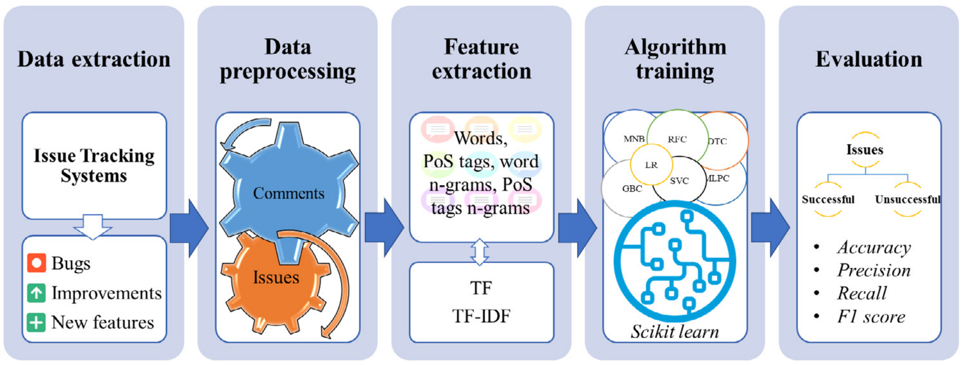


Fig. 3. Machine learning approach.

4. Research method

This work addresses the following questions.

RQ$_1$: Are textual descriptions of issues and comments from Jira ITSs useful to predict issue success in software projects?

RQ$_{1.1}$: What kind of information is useful to predict issue success?

RQ$_{1.2}$: How does the prediction of issue success vary over time?

RQ$_{1.3}$: How does the prediction of issue success vary with respect to bugs, improvements and new features?
The research questions can be answered using a machine learning approach. From the machine-learning perspective, the issue success prediction can be viewed as a binary-class classification problem, in which automatic methods have to assign positive or negative class labels (success or not success) to objects (texts).

In order to train a model that is able to perform predictions of issue success, a dataset for the learning process is required. With this aim, a set of comments and issues from ITSs were first collected. Texts were preprocessed to be easily understood by machine learning algorithms and to delete irrelevant information from them. Relevant features (characteristics) were extracted from the preprocessed texts, and the texts were transformed into a numerical representation in the form of a vector; then, machine learning algorithms were trained on the vectorized texts to produce prediction models, which were used to predict issue success. The general steps of the machine learning approach are shown in Fig. 3 and are detailed in the following sections.

4.1. Data extraction

The data extraction process was performed to collect the data that were required to train classification algorithms in order to perform predictions of issue success and answer the research questions. The following four public Jira ITSs store a considerable number of software projects (many of them open source) that are of the interest of many people dedicated to software engineering, so they were selected as data source.

- Apache's JIRA Issue Tracking System[1] is an open Issue Tracking System that stores more than 600 software projects.
- Atlassian's public Issue Tracking System[2] is used to manage more than 30 software projects of Confluence, Bitbucket, Jira and other Atlassian products.
- Red Hat's Issue Tracking System[3] stores more than 400 software projects including Red Hat projects.
- Spring's Issue Tracking System[4] is a system for tracking issues, progress, and roadmaps for more than 80 Spring projects and their derivatives.

With the aim of studying the success of different types of software tasks, the following three types of issues were extracted from the above Jira ITSs using the Jira API[5] and the web pages of the Jira ITSs.

- Bugs. Software defects and failures that affect software outcomes.
- Improvements. Software upgrades and enhancements.
- New features. New functionalities of software systems.

In order to study a sufficient number of issues of each type (at least 10K), issues of 588 software projects (333 from the Apache's ITS, 14 from the Atlassian's ITS, 191 from the Red Hat's ITS, and 50 from the Spring's ITS) were extracted. These projects represent about 50% of the projects stored in the mentioned repositories and were selected to study different software systems such as frameworks, software extensions, servers, libraries and web components. The selected projects vary on size and the number of people involved on them. Table 1 shows the distribution of projects according to their size (number of issues), and Table 2 shows the distribution of the projects according to their number of developers (assignees), commenters and watchers of the studied issues. Most of the projects are developed using Java and JavaScript as programming languages.

The issues of the selected projects are publicly visible, are mostly written in English, and were registered from April 2001 to September 2019. The status of the extracted issues is "Closed", which indicates that a resolution for each issue exists and that the issues are considered finished. The extracted issues are tagged according to their resolution (Fig. 2), so the issues that were resolved successfully and the issues that were resolved unsuccessfully were easily identified. Data of issues (textual description, issue type, date of creation, date of resolution, date of last update, status and resolution) and data of comments (id, textual description, id of the issue they belong and date of reporting) were stored in a local database with the aim of facilitating the preprocessing process. The complete dataset is available online (Ramírez-Mora et al., 2020).

Thirty thousand issues (15,000 successfully resolved and 15,000 unsuccessfully resolved) and their comments (about 120k) were particularly studied as a representative sample of the issues of the extracted projects. About 10,000 issues of each type (5000 successfully resolved and 5000 unsuccessfully resolved) were

1 Apache's ITS: https://issues.apache.org/jira/secure/Dashboard.jspa.
2 Atlassian's ITS: https://jira.atlassian.com/secure/BrowseProjects.jspa.
3 Red Hats's ITS: https://issues.redhat.com/secure/Dashboard.jspa.
4 Spring's ITS: https://jira.spring.io/secure/Dashboard.jspa.
5 JIRA Agile REST API Reference: https://docs.atlassian.com/jira-software/REST/7.0.4/

Table 1
Distribution of the studied projects by number of issues.

Projects by total number of issues

Repository	Number of issues			
	0–1000	1001–2000	2001–3000	More than 3000
Apache	204	40	29	60
Atlassian	3	2	0	9
Red Hat	123	25	15	28
Spring	35	9	1	5

Projects by number of studied issues[a]

Repository	Number of issues			
	0 - 500	501 - 1000	1001 - 1500	More than 1500
Apache	222	39	25	47
Atlassian	5	1	4	4
Red Hat	147	20	6	18
Spring	38	8	1	3

[a]Closed bugs, new features and improvements.

Table 2
Distribution of the studied projects by assignees, commenters and watchers considering the studied issues.

Projects by number of assignees

Repository	Number of assignees			
	0–200	201–400	401–600	More than 600
Apache	227	35	18	53
Atlassian	13	1	0	0
Red Hat	191	0	0	0
Spring	50	0	0	0

Projects by number of commenters

Repository	Number of commenters			
	0–200	201–400	401–600	More than 600
Apache	199	57	28	49
Atlassian	3	1	0	10
Red Hat	143	23	10	15
Spring	36	6	4	4

Projects by number of watchers

Repository	Number of watchers			
	0–1000	1001–2000	2001–3000	More than 3000
Apache	258	25	12	38
Atlassian	4	1	4	5
Red Hat	160	12	9	10
Spring	45	2	0	3

selected, so the dataset is balanced with respect to both, issue type and issue successfulness. The detailed distribution of the studied issues and comments is shown in Table 3. The balanced dataset was selected with the aim of studying a sufficient number of successful and unsuccessful issues of each type: the extracted software projects have, in general, more successful issues than unsuccessful issues (only 10%–20% of issues that are resolved in less than 30 days are unsuccessful) and most of the issues are bugs (improvements and new features represent a very small part of the total number of issues).

In addition to the selected issues in Table 3, the project with the greatest number of issues of each repository was selected to be studied in detail. This was performed with the aim of studying prediction of issue success in particular real projects. The selected projects are shown in Table 4.

4.2. Preprocessing

The following tasks were performed to eliminate irrelevant information from issues and comments after the extraction process. In addition, the preprocessing aims at reducing the amount of information to be processed and improving prediction tasks.

- URLs were replaced with the string "url_specification".
- Specific user names were identified and replaced with the string "user_specification".
- Some comments included the string "+1", which means that the reporter of the comment expressed a positive vote to resolve the related issue, so this string was replaced with "vote_specification".
- Numbers and software versions that were identified in the texts were replaced with "number_specification" and "version_specification" respectively.
- Specific emails were replaced with the string "email_specification".
- Some comments included fragments of software code, which were replaced with the string "code_specification".
- Some comments included paths to specify file directories, which were replaced with the string "path_specification".

The preprocessed issues and their comments were tagged as "successful" or "unsuccessful" according to the resolution of each issue (Fig. 2). This tagging was required to perform supervised predictions using machine learning classifiers, which require a set of texts labeled with the class they belong. Issue resolution time (in days) was calculated by subtracting the date of creation to the date of resolution of each issue; the number of days between the creation of an issue and the registration of each of their

Table 3
Distribution of the studied issues and comments.

Issue type	Successful issues		Unsuccessful issues	
	Issues	Comments	Issues	Comments
Bugs	≈ 5000	≈ 25 000	≈ 5000	≈ 17 000
Improvements	≈ 5000	≈ 21 000	≈ 5000	≈ 19 000
New features	≈ 5000	≈ 22 000	≈ 5000	≈ 19 000
Total	≈ 15 000	≈ 68 000	≈ 15 000	≈ 55 000

Table 4
Projects selected to be studied in detail.

	Project A	Project B	Project C	Project D
Key	FLEX	JSWSERVER	JBIDE	SPR
Repository	Apache	Atlassian	Red Hat	Spring
Issues (total)	35 382	12 559	26 110	17 413
Studied issues[a]	18 209	3076	13 122	8714
Comments[a]	55 014	7210	72 754	34 925
Watchers[a]	490	11 248	25 239	15 916
Assignees[a]	138	109	108	39
Commenters[a]	143	2194	1189	3136

[a]Closed bugs, new features and improvements.

comments was also calculated. These data were calculated to perform predictions of issue success considering periods of time.

4.3. Feature extraction

Textual descriptions of comments and issues (which have lexical, syntactic and semantic features) are used in this work to perform prediction tasks. Sequences (n-grams) of words are usually used as lexical features. Syntactic features relate to the structure of texts. Part of Speech (PoS) tags are morphological features that are used to categorize a word in accordance with its syntactic function (nouns, adjectives, verbs, etc.), so they are often used to define syntactic features. Semantic features relate to the meaning of words in a text. The following features were selected to be extracted from the preprocessed descriptions of issues and comments.

- Word n-grams. Words and sequences of words varying from 2 to 10 words were considered.
- PoS tags n-grams. PoS tags and sequences of PoS tags varying from 2 to 5 were considered.

When texts are required to construct models that are used by machine learning classifiers to perform prediction tasks, they must be transformed into numerical vectors that can be understood by the classifiers. Text vectorization is performed based on defined features and weighting schemes. The following weighting schemes were selected to vectorize the descriptions of issues and comments.

- Term frequency (TF). Score representing the number or occurrences of a term in a document (Jones, 1972).
- Term frequency–Inverse document frequency (TF–IDF). Numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus (Rajaraman and Ullman, 2011; Wu et al., 2008).

Experiments were performed using the TF weighting scheme, and then, experiments were also conducted using the TF–IDF weighting scheme. This was performed with the aim of comparing which weighting scheme provided the best results. In both types of experiments, the defined features were considered, so the number of occurrences and the importance of each word n-gram and each PoS tag n-gram were used to perform predictions of issue success.

The feature extraction process was performed using the Python programming language and the scikit-learn library for Python (Manning and Schütze, 1999; Pedregosa et al., 2011). The *CountVectorizer* functionality of the scikit-learn library was used to convert the texts (descriptions of comments and issues) to a matrix of token counts considering the TF weighting scheme and the defined features. The *TfidfVectorizer* functionality was also used to convert the texts to a matrix of TF–IDF features.

4.4. Training

4.4.1. Algorithm selection

The classifiers are machine learning algorithms that are used to predict the classes of given objects. The following algorithms were selected to predict successful and unsuccessful issues.

- Multinomial Naïve Bayes (**MNB**). It is a Naïve Bayes algorithm variant that can be used to classify texts.
- Logistic Regression (**LR**). It is a linear model for classification in which the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.
- Support Vector Classifier (**SVC**). It is a Support Vector Machine (SVM), which is a non-probabilistic binary linear classifier.
- Decision Tree Classifier (**DTC**). It is a tree-like model used to perform classifications.
- MLP Classifier (**MLPC**). It is an Artificial Neural Network (ANN) model to perform predictions.
- Random Forest Classifier (**RFC**). It is an ensemble method that uses various decision tree classifiers.
- Gradient Boosting Classifier (**GBC**). It is an ensemble method that supports both binary and multi-class classification.

The above algorithms are implemented in the scikit-learn library for Python (Pedregosa et al., 2011). The algorithms were selected to identify those with the best results and to compare the results of performing predictions of issues success with results of related works that reported the use of those classifiers (the works of Guo et al. (2010) and Shippey et al. (2019) reported the use of LR classifier, the works of Di Sorbo et al. (2019) and Menzies et al. (2007) reported the use of Naïve Bayes classifier, and Hall et al. (2012) reported the use of Naïve Bayes and LR classifiers). In addition to the classifiers used in related works, ensemble methods (RFC, which is an averaging method, and GBC, which is a boosting method) were selected to compare results with results of non-ensemble methods such as DTC. Probabilistic classifiers (such as

Table 5
Summary of variables for the designed experiments.

Issue types (T)	Algorithms (A)	Periods of time in days (N)	Features (F)	Weighting schemes (W)
Bugs, Improvements and New features	MNB, LR, SVC, DTC, MLPC, RFC, GBC	1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350,..., 3500. A total of 80 periods of time were considered.	Word n-grams, PoS tag n-grams	TF, TF–IDF

MNB) and non-probabilistic methods (such as SVC) were selected to determine which type of classifiers performed better. A Multi-Layer Classifier (MLPC) was also selected due to its capability to learn non-linear models and its capability to learn models in real-time (Pedregosa et al., 2011).

A detailed description of the above algorithms and the parameters that were used to perform experiments are presented in Appendix A.

4.4.2. *Description of experiments*

Experiments were designed to answer the research questions regarding the usefulness of texts (descriptions of issues and comments) to predict issue success. The experiments were performed using the previously selected features, weighting schemes and machine learning algorithms. Experiments were also performed by issue type (bugs, improvements and new features) with the aim of analyzing the differences when predicting the success of each issue type. The prediction of issue success over time is studied in this work, so experiments were designed considering the issue resolution time and the time between the creation of issues and the date of reporting of their comments. The resolution time of most of the issues varied from 1 to 3500 days, so periods of time in this interval were considered. Table 5 shows a summary of the variables that were used to perform the designed experiments. In each experiment, predictions of issue success were performed using an algorithm (A), feature (F), weighting scheme (W), descriptions of issues of a specific type (T) that were resolved in more than a specific number of days (N), and descriptions of comments that were registered in the first (N) days since the creation of their respective issues.

A total of 6720 experiments were designed and conducted (combining three issue types, seven classifiers, 80 periods of time, two weighting schemes and two feature types) using the selected issues and comments in Table 3. For each experiment, the following steps were performed using Python and the scikit-learn library (Pedregosa et al., 2011).

- The description of each issue and the descriptions of its comments were joint into a single text. This was performed because several comments of an issue can provide more information about the progress of the issue than individual comments and, thus, improve the prediction of issue success.
- Feature extraction was performed to vectorize the selected texts considering the defined features and weighting schemes.
- The seven algorithms in Section 4.4.1 were trained on the vectorized texts to produce prediction models. The texts were divided as follows: 75% was included in the training set and 25% in the test set, which was used to evaluate the results. When the number of successful issues was greater than the number of unsuccessful issues (or vice versa), the training set was balanced to perform experiments with the same number of texts in each class using under-sampling balancing (reducing the size of the biggest class). The training set was balanced to represent both classes equally. Experiments with balanced test sets were performed to calculate the accuracy measure (which is sensitive to unbalanced classes), and then, experiments with unbalanced test sets were performed to calculate precision, recall and F1-score in a realistic way (considering the real distribution of issues by periods of time).

The above steps were also performed considering the issues of projects in Table 4. These experiments were performed considering a balanced training set and an unbalanced test set.

4.5. *Evaluation*

The most used metrics to evaluate the results of machine learning algorithms are based on the number of “True” (correctly predicted) and “False” (incorrectly predicted) values of two generic classes: “Positive” and “Negative”. According to the issue resolution tags (Fig. 2), issues (bugs, improvements and new features) are categorized in two general classes (successful and unsuccessful), which can correspond to the “Positive” and “Negative” classes respectively. Based on this, the values that can be used to measure the performance of machine learning algorithms are: True Positives (number of correctly predicted successful issues), False Positives (number of incorrectly predicted successful issues), True Negatives (number of correctly predicted unsuccessful issues), and False Negatives (number of incorrectly predicted unsuccessful issues). These values are summarized in the confusion matrix in Table B.8 (Appendix B). Precision, accuracy, recall and F1 score (usually calculated to evaluate the performance of machine learning classifiers) were calculated in each experiment and are described in Appendix B.

5. Results

In general, PoS tag n-grams were not useful to predict issue success, and the best results were obtained using the TF–IDF weighting scheme and word n-grams as features. The results of predicting issue success considering three issue types, 80 periods of time, seven classifiers, the weighting scheme TF–IDF, and word n-grams are described in detail in the following sections.

5.1. *Prediction of issue success*

Accuracy of predictions of issue success varied for each issue type (bugs, improvements and new features): the accuracy was from 0.38 to 1.0 in experiments with bugs, from 0.25 to 0.83 in experiments with improvements and from 0.22 to 1.0 in experiments with new features. The predictions of bug success were more accurate than the predictions of the success of improvements and new features. The most accurate algorithms were LR, MNB, GBC and MLPC, and the least accurate was SVC. Fig. 4 graphics the accuracy measurements of predicting issue success by issue type. Each of the 21 box plots in Fig. 4 summarizes the accuracy measurements of 80 experiments (corresponding to 80 periods of time) and shows the minimum, maximum, first quartile, third quartile, median and mean of the accuracy measurements that were achieved for a specific machine learning

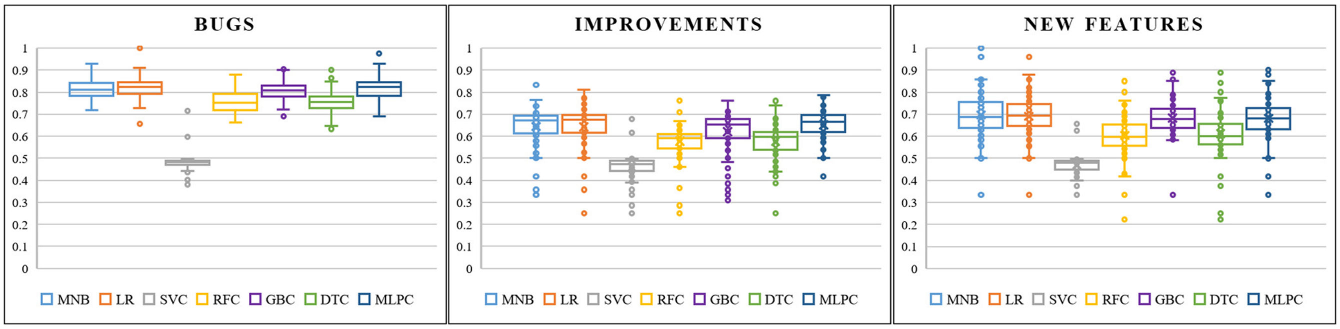


Fig. 4. Accuracy measurements of predictions of issue success by issue type and classifier.

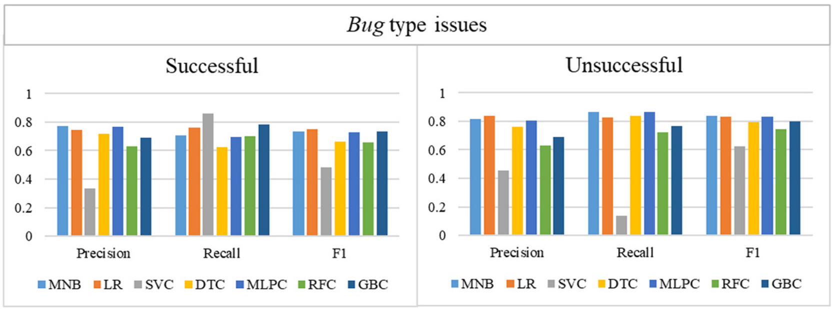


Fig. 5. Average performance values of predictions of successful and unsuccessful issues (bug type).

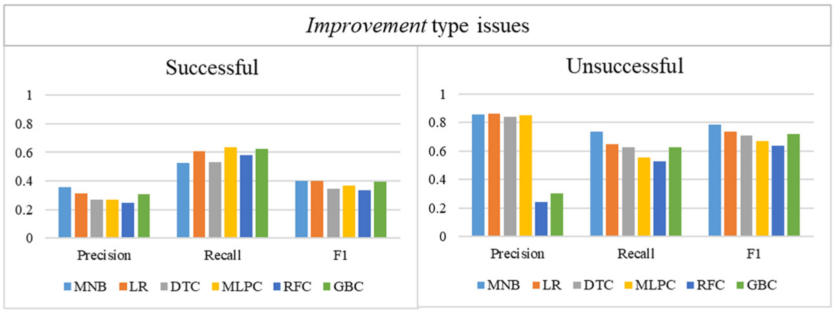


Fig. 6. Average performance values of predictions of successful and unsuccessful issues (improvement type).

classifier. Descriptive statistics of accuracy measurements are detailed in Table C.9 (Appendix C).

The precision, recall and F1 score of predicting issue success varied from 0.0 to 1.0 for all the issue types (bugs, improvements and new features). The mean precision of predicting unsuccessful bugs and the mean precision of predicting successful bugs were similar. Figs. 5–7 show the mean performance results of predicting issue success by issue type and classifier. The mean performance measurements (precision, recall and F1) of predicting successful and unsuccessful bugs were higher than the mean performance measurements for predicting improvements and new features. In most of the experiments, SVC achieved the worst results and in most of experiments with improvements and new features, SVC did not predict any successful or unsuccessful issue, so its performance measurements were not included. Tables C.9–C.12 in Appendix C present the maximum, minimum, mean, variance and standard deviation of the performance measurements (accuracy, precision, recall and F1 score) by issue type and classifier, and each value was calculated from the results of 80 experiments using the data in Table 3.

In general, the best classifiers were MNB, LR, GBC and MLPC (which results were statistically similar) followed by RFC and DTC. The worst classifier was SVC and its results were statistically different to the results of the other classifiers. The performance of MNB and LR classifiers were the best (they performed predictions in less than one second), followed by RFC, DTC and MLPC (they performed predictions in less than one minute), and GBC (with less than two minutes to perform predictions). SVC performed predictions in about nine minutes, so its performance was the worst. The performance (time to perform predictions) of the classifiers was measured in an Intel® CORE i5 7th generation processor in a Windows 10 (64 bits) system.

5.2. Variation of predictions of issue success over time

Predictions of issue success were performed considering 80 measures of time in days (Table 5), which indicate the first *N* days since an issue was created. For each period of time, experiments were performed considering only comments that were created in such period of time and issues that had not been resolved yet. Figs. 8–10 show the accuracy for predicting issue success considering the time measures and seven machine learning classifiers. The accuracy of predicting bug success (Fig. 8) tended to increase as larger periods of time were considered. This is represented by the trend line in Fig. 8. The increment on the accuracy of most of the classifiers (except for the SVC) was particularly clear

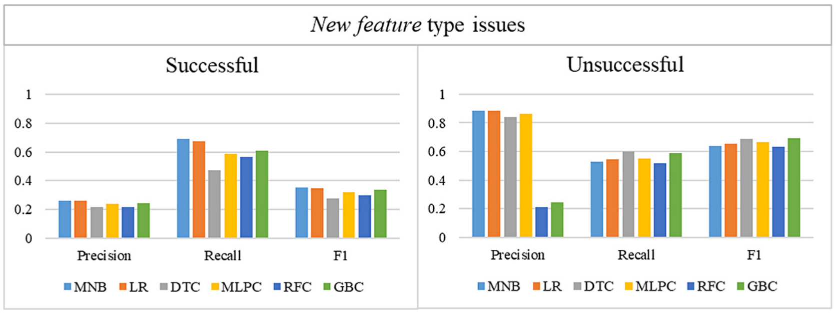


Fig. 7. Average performance values of predictions of successful and unsuccessful issues (new feature type).

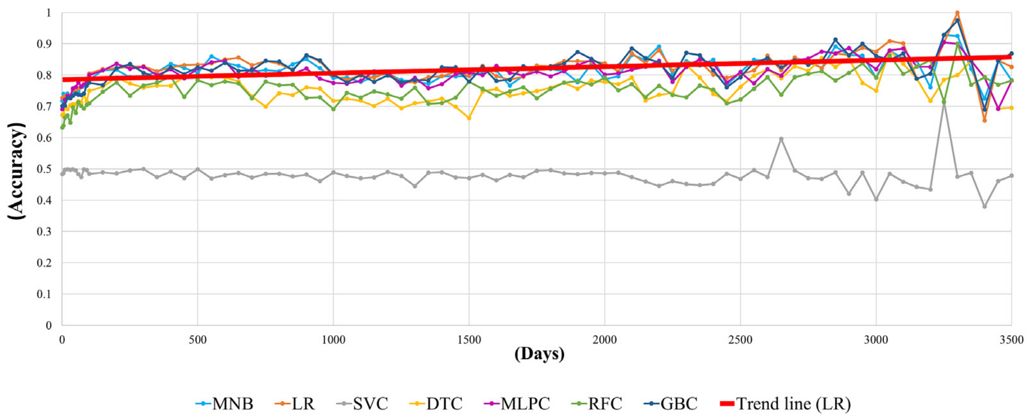


Fig. 8. Accuracy for predicting bug success over time.

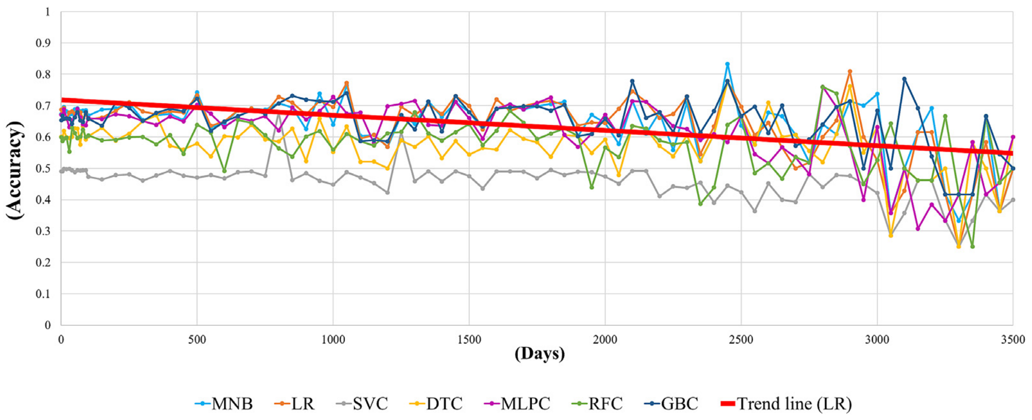


Fig. 9. Accuracy for predicting improvement success over time.

in the first 500 days. The accuracy of predicting the success of improvements (Fig. 9) seemed to increase until 500 days, after greater periods of time, the accuracy tended to decrease. This decrement is represented by the trend line in Fig. 9. The accuracy of predicting the success of new features seemed to be little variant in the first measures of time, and after 1000 days, it seemed to be irregular; however, the accuracy of predicting the success of new features tended to increase as the number of days that were considered increased (Fig. 10). In general, the accuracy of the classifiers seemed to be variant when periods of time of more than 1500 days were considered.

Precision, recall and f1 score were calculated in experiments by periods of time using data of individual projects in Table 4. These results were compared with the results using the general dataset in Table 3. Results showed that precision and recall of experiments with the general dataset were better than results of experiments with most of the individual projects, as shown in Figs. 11 and 12. Precision of predicting successful bugs, and recall of predicting unsuccessful bugs using issues and comments of Project C were very similar to the results of predictions using the general dataset; however, recall and precision could vary depending on the distribution and number of issues and comments.

Precision and recall of predicting bug success using data from Project A were less variant than results of experiments using data from the other projects. This is explained because project A has more issues and comments than the other projects, and predictions were less variant. Most of the predictions of successful bugs using data from specific projects achieved more than 70% of precision in the first measures of time. Precision of predicting unsuccessful bugs, and recall of predicting successful and unsuccessful bugs tended to increase as larger periods of time

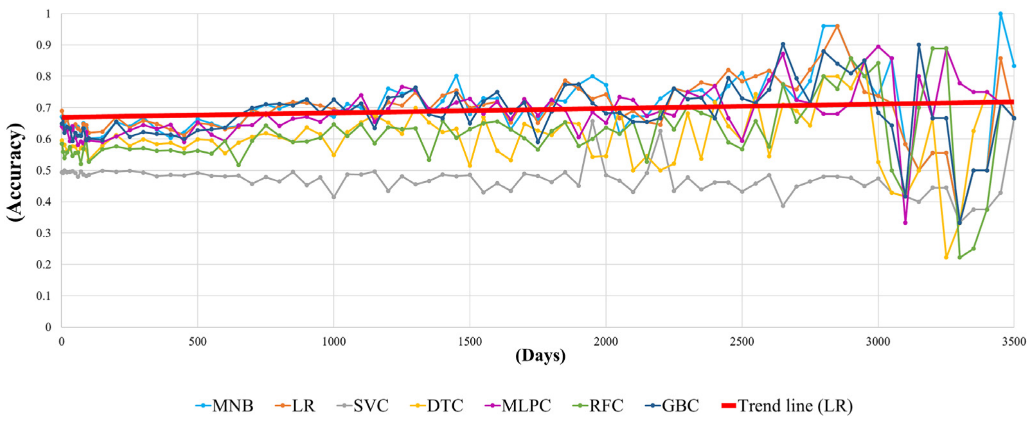


Fig. 10. Accuracy for predicting new feature success over time.

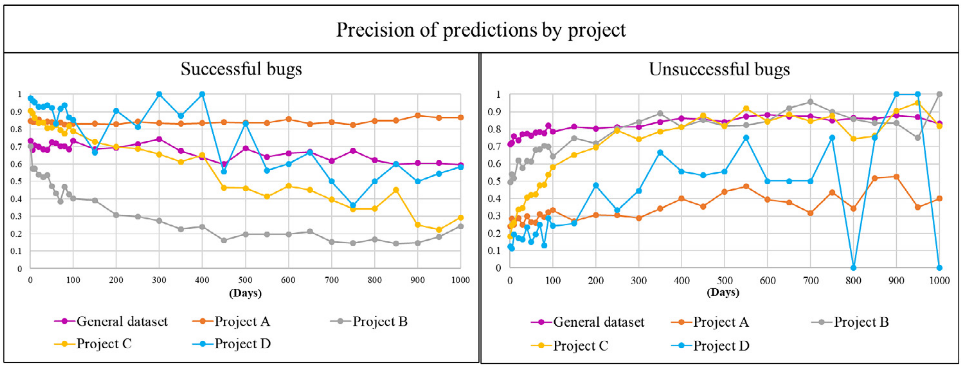


Fig. 11. Precision for predicting bug success over time by project using LR classifier.

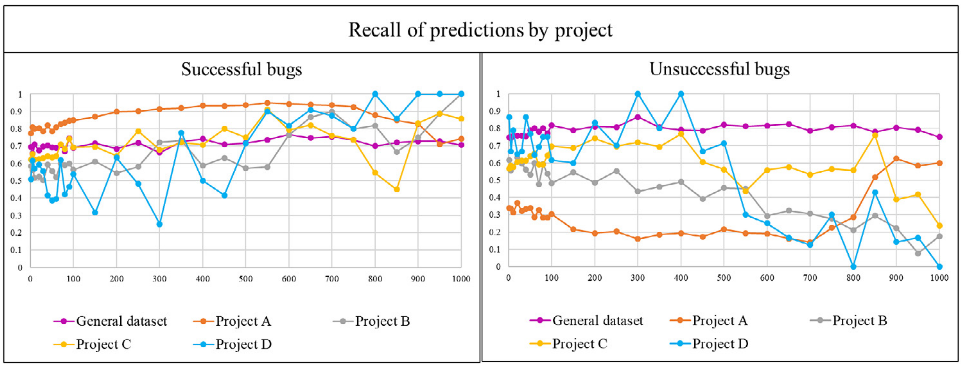


Fig. 12. Recall for predicting bug success over time by project using LR classifier.

were considered using both, the general dataset and data from the specific projects.

As shown in Fig. 13, the percentage of successful issues decreases as higher periods of time are considered. Correlations between the percentage of successful issues and time were calculated by project using the Pearson's correlation coefficient (Pearson, 1895; Stigler, 1989). The correlation coefficient was −.936 for project A, −.88 for project B, −.98 for project C and −.56 for project D. These correlations indicate that the development of software tasks was more effective in the first periods of time and that more unsuccessful issues existed at the end of the projects.

5.3. Relevant information for predicting issue success

The most relevant features (characteristics of descriptions of issues and comments) for predicting issue success were identified in each of the 80 experiments by issue type using the general dataset. The features were ranked according to their importance in each experiment according to their weights (numeric values that were obtained in the vectorization process). Tables 6 and 7 show the most relevant features and indicate the times the features were among the 100 most relevant features and the average ranking of each feature. Only the relevant features for predicting issue success that were identified in experiments using the general dataset and that were also identified in experiments using data of the specific projects were included in Tables 6 and 7. In general, the most relevant features for predicting issue success were unigrams of words.

Most of the relevant features in Tables 6 and 7 are nouns, and most of them are technical concepts that relate to the context of the study (software development). The most relevant phrases for predicting issue success were also identified. Figs. 14 and 15 show

Table 6
Most relevant features for predicting successful issues by issue type.

#	Bugs			Improvements			New features		
	Feature	N	AR	Feature	N	AR	Feature	N	AR
1	patch	80	26.97	currently	78	34.92	used	79	26.81
2	file	80	36.36	property	75	34.42	message	76	27.77
3	problem	80	55.61	added	74	43.46	make	76	52.42
4	fixed	79	30.31	created	74	64.92	set	75	55.14
5	bug	79	47.73	methods	71	47.74	security	75	37.35
6	document_type	78	48.27	class	69	51.54	attribute	72	63.73
7	attachment	70	54.25	exception	69	54.51	component	66	57.52
8	code	70	82.33	build	69	66.87	implement	65	89.02
9	thanks	67	72.46	allow	66	69.96	web	65	98.81
10	line	65	83.52	object	64	61.91	based	60	136.97

N: Number of experiments in which the feature was among the 100 most relevant features. AR: Average ranking of the feature considering 80 experiments.

Table 7
Most relevant features for predicting unsuccessful issues by issue type.

#	Bugs			Improvements			New features		
	Feature	N	AR	Feature	N	AR	Feature	N	AR
1	url_spec.	80	3.27	url_spec.	80	2.08	code_spec.	80	4.32
2	review	80	26.71	version_spec.	80	6.96	like	80	8.68
3	project	80	30.91	code	80	12.43	need	80	14.32
4	error	80	38.21	make	80	18.91	project	80	17.52
5	issue	78	44.62	need	80	20.23	using	80	22.61
6	test	75	62.72	using	80	20.25	patch	80	25.66
7	user	70	91.62	issue	80	20.28	feature	80	29.08
8	click	69	64.92	new	80	20.60	just	80	29.43
9	spring	66	66.08	project	80	23.58	version	80	30.07
10	would	60	97.62	add	80	23.67	nice	79	38.42

N: Number of experiments in which the feature was among the 100 most relevant features. AR: Average ranking of the feature considering 80 experiments.

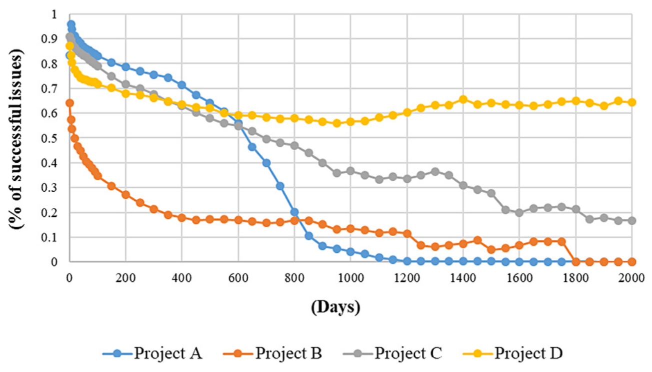


Fig. 13. Percentage of successful issues over time by project.

10 of the most representative and relevant phrases for predicting issue success by issue type. The phrases were classified according to their function (or purpose) with the aim of understanding the communication functions associated with issue success in software development. The phrases in Figs. 14 and 15 were classified according to the following categories of communication functions proposed by Jakobson (1963): the referential function indicates that a phrase describes a situation, context, or state; the conative function is used in imperative sentences and indicates that a phrase is intended to change people behavior; and the emotive function indicates that a sentence expresses emotions, thoughts, wishes, etc. Most of the relevant phrases for predicting successful issues were referential phrases, particularly for predicting successful bugs and improvements (Fig. 14). The purpose of many of these phrases was to provide technical information. Some conative phrases and few emotive phrases were identified as useful for predicting successful issues. Most of the relevant phrases for predicting unsuccessful issues were emotive phrases, which are not intended to provide technical information. Few referential phrases, and only one conative phrase, were identified as relevant phrases for predicting unsuccessful issues (Fig. 15).

PoS tags n-grams were not identified as relevant features for the prediction of issue success; however, some of the most representative PoS tags n-grams were identified and are the following: “noun + noun”, “determiner + noun”, “noun + preposition”, and “noun + verb”.

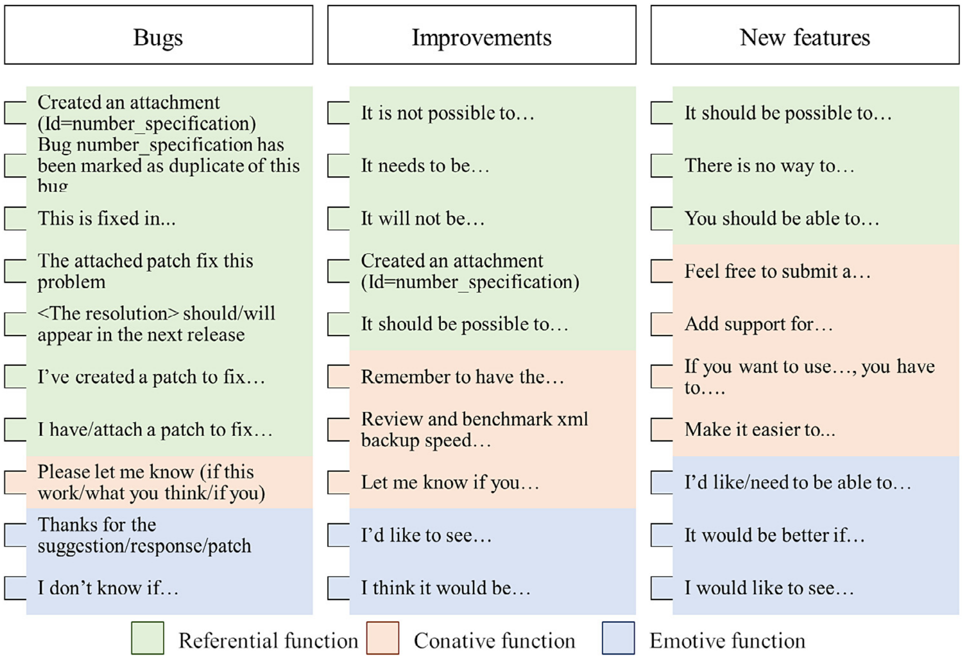
6. Discussion

6.1. Answers to research questions

6.1.1. Are textual descriptions of issues and comments from Jira ITSs useful to predict issue success in software projects?

In general, descriptions of issues and comments were useful for predicting the success and failure of issues, and particularly, descriptions of bugs and their comments were very useful. The best classifiers achieved an accuracy and precision between 80% and 100% for predicting issue success in many experiments. Since day one, some classifiers achieved an accuracy and precision of more than 70% for predicting successful and unsuccessful bugs, which indicates that the success or failure of more than 70% of bugs can be predicted correctly the day the bugs are reported using their descriptions and comments. Since day 30, some classifiers achieved more than 80% of precision and recall when predicted bug success.

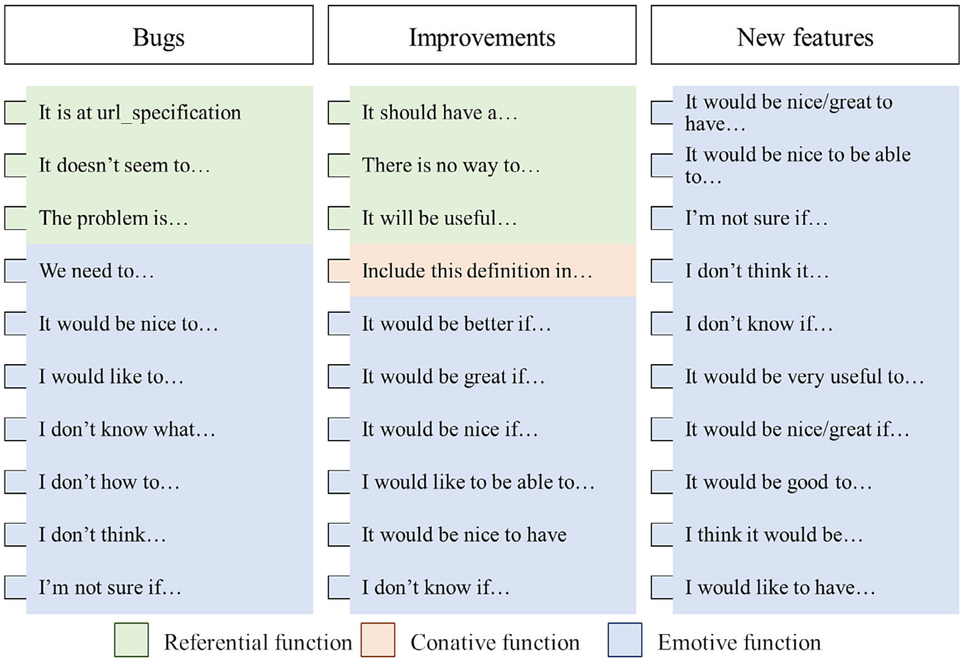
The evaluation of prediction results depends on the type of the prediction problem. In this work, the prediction of the success or failure of issues is aimed to make decisions early and take actions to ensure the successful completion of software tasks. If it is predicted that an issue will be unsuccessfully performed, actions to ensure its success can be taken, and if the prediction is wrong, extra actions to ensure the successfully completion of the issue do not affect software quality. Based on the above, more



Bugs	Improvements	New features
🟩 Created an attachment (Id=number_specification)	🟩 It is not possible to…	🟩 It should be possible to…
🟩 Bug number_specification has been marked as duplicate of this bug	🟩 It needs to be…	🟩 There is no way to…
🟩 This is fixed in...	🟩 It will not be…	🟩 You should be able to…
🟩 The attached patch fix this problem	🟩 Created an attachment (Id=number_specification)	🟧 Feel free to submit a…
🟩 <The resolution> should/will appear in the next release	🟩 It should be possible to…	🟧 Add support for…
🟩 I've created a patch to fix…	🟧 Remember to have the…	🟧 If you want to use…, you have to….
🟩 I have/attach a patch to fix…	🟧 Review and benchmark xml backup speed…	🟧 Make it easier to...
🟧 Please let me know (if this work/what you think/if you)	🟧 Let me know if you…	🟦 I'd like/need to be able to…
🟦 Thanks for the suggestion/response/patch	🟦 I'd like to see…	🟦 It would be better if…
🟦 I don't know if…	🟦 I think it would be…	🟦 I would like to see…

🟩 Referential function 🟧 Conative function 🟦 Emotive function

Fig. 14. Most relevant phrases for predicting successful issues.



Bugs	Improvements	New features
🟩 It is at url_specification	🟩 It should have a…	🟦 It would be nice/great to have…
🟩 It doesn't seem to…	🟩 There is no way to…	🟦 It would be nice to be able to…
🟩 The problem is…	🟩 It will be useful…	🟦 I'm not sure if…
🟦 We need to…	🟧 Include this definition in…	🟦 I don't think it…
🟦 It would be nice to…	🟦 It would be better if…	🟦 I don't know if…
🟦 I would like to…	🟦 It would be great if…	🟦 It would be very useful to…
🟦 I don't know what…	🟦 It would be nice if…	🟦 It would be nice/great if…
🟦 I don't how to…	🟦 I would like to be able to…	🟦 It would be good to…
🟦 I don't think…	🟦 It would be nice to have	🟦 I think it would be…
🟦 I'm not sure if…	🟦 I don't know if…	🟦 I would like to have…

🟩 Referential function 🟧 Conative function 🟦 Emotive function

Fig. 15. Most relevant phrases for predicting unsuccessful issues.

than 80% of precision and recall of predicting issue success can be considered as good because at least 80% of the issues that will be unsuccessfully performed can be identified after 30 days and can be managed to avoid their failure, and at least 70% of the issues that will be unsuccessfully resolved can be predicted since the day of their report. The time for resolving an issue is 500 days on average, so the success or failure of most of the issues can be predicted with more than 80% of precision in the 10% of the time that takes their resolution. This indicates that, since the early activities involved in an issue resolution, early decisions can be taken and actions can be performed to ensure the success of the issue.

6.1.2. *What kind of information is useful to predict issue success?*

The most relevant features for the prediction of issue success were single words related to software development processes. The words “added”, “created”, “fixed” and “used” were relevant words for predicting successful bugs, improvements and new features. These words were used in phrases to report an action intended to resolve issues.

The relevant words for predicting unsuccessful issues included “url_specification” and “code_specification”, which refer to URLs and code fragments that were used to indicate that specific software modules or features needed to be fixed. The words “project”, "error" and "review" were also relevant for predicting unsuccessful issues. These words were used to report specific software concerns related to issues that were unsuccessfully addressed.

The phrases that were more relevant for predicting issue success were referential phrases. Phrases that provided technical information, described situations, and reported work progress, were strongly related to successful bugs. This indicates that during the correction of bugs that will be successfully resolved, people share technical information that is useful for the correction of bugs (such as attachments, patches, resolutions and code) and report advances on the correction of bugs, which highlights objective communication.

In general, most of the relevant phrases for predicting successful issues were referential and conative phrases. This suggest that when issues are being addressed successfully, people report work, describe context, suggest work and exhort people to perform work through comments. In contrast, the most relevant phrases for predicting unsuccessful issues were emotive phrases, which express feelings, emotions and thoughts. This indicates that when issues that will be unsuccessfully attended are being developed, people express personal aspects and concerns about work that is not being developed rather than technical information. The phrases "I don't know…" and "I'm not sure…", which were identified as relevant phrases for predicting unsuccessful issues, indicate misunderstanding and unclarity. This highlights that an ineffective communication may relate to unsuccessful issues because effective communication implies clarity and that the intended meanings equals the perceived meanings (Schermerhorn et al., 2002).

Some referential phrases such as "The problem is…" or "There is no way to…", were also identified as relevant phrases for predicting unsuccessful issues, but they indicated that something was going wrong, and that few advances on issue resolution existed.

6.1.3. *How does the prediction of issue success vary over time?*

Predictions of issue success were performed considering 80 periods of time. In general, accuracy and some precision and recall measures of issue success predictions tended to increase as larger periods of time were considered. Predictions in first periods of time are most relevant because people can know early whether issues will be successfully resolved or not. Accuracy of predicting improvement success tended to increase until 550 days, and after that, it tended to be irregular and decreased in the last periods of time. This indicates that, in general, the information related to bugs and new features is very useful for predictions in most of the periods of time, and that the first comments that were registered during the development of improvements are the most useful for predicting issue success. This highlights that the information communicated during the resolution of bugs and new features is related to the success or failure of issues during all the resolution process, and that only the initial information that is communicated during the development of improvements is related to the resolution result. Improvements are perfective maintenance tasks, and about 86% of their comments are written before day 500, and after that, new useful information (such as referential phrases) rarely arises. This indicates that after day 500, few information is provided because few issues remain unresolved. This causes that accuracy and precision of predictions decrease after day 500.

Accuracy of experiments for predicting the success of three types of issues (bugs, new features and improvements) was less variant considering the first measures of time than the accuracy of experiments considering the last measures. This can be explained because the number of texts decreased in the last measures of time (in the experiments, issues that have not been resolved yet were considered, so in experiments considering the initial measures of time, predictions were performed with more texts than in the last predictions when periods of time of more than 2000 days where considered), and few texts could make the results of predictions unstable and variable.

Bugs, which are activities of corrective maintenance, were attended in an approximate time of 480 days, improvements in 460 days and new features in 550 days. This indicates that an accurate prediction of bug success and improvement success can be performed earlier than an accurate prediction of the success of new features because the progress on the resolution of bugs and improvements can be reflected in comments earlier than in comments of new features.

6.1.4. *How does the prediction of issue success vary with respect to the three issue types (bugs, improvements and new features)?*

In general, predictions of bug success were more accurate and precise than the predictions of the success of improvements and new features. Bugs are relevant software tasks because affect software outcomes, so when a defect or failure is identified, people must begin working on fixing it. Bugs are fixed in less time than new features, so an important number of comments of bugs that are reported in the first periods of time are useful for predicting bug success. This may indicate that the communication is more efficient when bugs are attended because their comments include important referential information for resolving them.

Improvements and new features are perfective and adaptive maintenance tasks respectively, which are not often a priority in software development, so few people are sometimes assigned to attend them. Results show that messages tend to be conative and poorly referential when improvements and new features are developed. This may explain that the messages that are communicated during the development of new features and improvements are less useful to perform predictions of issue success than the messages communicated during bug resolution.

6.2. *Related work*

In the present work, some of the classifiers that achieved the best results were Multinomial Naïve Bayes (which is a variation of Naïve Bayes) and Logistic Regression (LR). Other works that focus on predicting other software development outcomes (such as fault prediction), have found that these two algorithms performed well; however, almost no works on predicting issue success have been conducted. Menzies et al. (2007) showed that Naïve Bayes provided better performance for fault prediction than other algorithms. The results of Hall et al. (2012) showed that he models that performed well tend to be based on simple modeling techniques such as Naive Bayes or Logistic Regression. Shippey et al. (2019) identified code features for software defect prediction and found that Logistic Regression achieved good results.

Some works highlighted the importance of comments for predictions in software projects. In the work of Choetkiertikul et al. (2018), the number of comments was one of the most relevant features. Valdivia-Garcia et al. (2018) concluded that the description and comments of bugs were the most important factors to predict bugs that block the fixing of other bugs. The results of Di Sorbo et al. (2019) demonstrated that it is possible to predict whether an issue will be closed as a "won't fix" using textual features from titles and descriptions of issues; and that Naïve Bayes had a precision of 0.795. In the present work, the mean precision for predicting bug success was over 0.80. The work of Di Sorbo et al. (2019) is similar to the present work; however, the present work studies three issue types and many types of issue resolutions classified in successful and unsuccessful. In the present work, descriptions of comments were also used to predict issue success (not only titles and descriptions of issues), and relevant phrases for predicting issue success (associated to communication functions) were identified. The present work studied issues from Jira ITSs and one of its most important contributions is the performing of predictions considering periods of time to study how predictions vary according to how long an issue has been opened. The present work provides evidence that prediction of issue success can be performed using data of opened (or not resolved yet) issues. In addition, the present work also contributes on studying the usefulness of seven machine learning classifiers to perform issue success predictions.

In the work of Guo et al. (2010), 68% of precision and 64% of recall were achieved when predicting Windows 7 bug fixes; in the present work (using only descriptions of issues and comments) the mean precision and recall of the best classifiers were greater than 80%. Guo et al. (2010) recommended to train employees to write high-quality bug reports and improve communication and trust amongst people. The present work provides evidence on the importance of communicated comments for issue success.

The results of Murgia et al. (2014) showed that perfective maintenance (including the development of improvements) is on average faster than corrective maintenance (including the resolution of bugs); and that corrective maintenance is on average faster than adaptive maintenance tasks (including the attention to new features). The results of the present work confirm the results of Murgia et al. (2014) because it was found that improvements are resolved in an average time of 460, bugs in 480 days and new features in 550 days.

6.3. Implications for practice

This study benefits the software development community because describes a simple way to predict the success of software tasks based on the analysis of texts (descriptions of issues and comments). Particularly, organizations that use ITSs to manage their projects can perform early predictions of issue success, reduce time in the development processes, increase software quality and customer satisfaction, and improve the organizational productivity. Early predictions of issue success can help project managers to distribute resources and attend critical issues that are likely to fail; developers can reduce development time because they could know early whether issues will be attended and take actions without having to wait until the resolution of a tasks to perform other software development activities; and customers can know early if their requirements will be satisfied.

This work provides evidence on how to use texts (descriptions of issues and comments) to predict issue success and evidence about the best machine learning classifiers to perform this task. This work highlights the most useful features and kind of texts to perform predictions of issue success with good accuracy and precision considering periods of time with the aim of helping organizations to perform accurate and precise predictions.

This work provides evidence that aspects of organizational performance (such as issue success) can be predicted using textual information about interactions among people (not always, the use of technical information is required), so organizations can perform predictions in a simple way. Organizations dedicated to software development must consider a frequent analysis of data from ITSs as an important strategy for improving software development tasks.

Based on the results, authors of the present work recommend the following: include referential information on descriptions of issues and comments to increase the quality of texts and the probability of issue success; encourage objective, frequent and open communication among people involved in a software project; perform predictions of issue success as soon as possible (enough information can be obtained to perform highly accurate and precise predictions after 30 days an issue was registered); implement an automatic process to predict issue success that can be updated continuously with recent data to perform predictions; use MNB and LR when texts are considered to predict issue success due to their precise and accurate results and the time they require to perform predictions.

6.4. Threats to validity

In this section, the threats that have an impact on the validity of the results are discussed, including threats to construct validity, threats to internal validity, threats to external validity, threats to conclusion validity, and threats to reliability.

Construct validity refers to the degree to which the operationalization of the measurements in a study actually represents the constructs in the real world (Jedlitschka et al., 2008). In this work, the studied data (issues and comments of real software projects) were directly extracted from four Jira ITSs and were not manipulated before being processed, so the data represent information of the real world. One of the advantages of using data from electronic databases such as ITSs, is that the extracted data is stable and is not influenced by the presence of researchers (Jedlitschka et al., 2008).

Internal validity refers to the extent to which the treatment or independent variable(s) were actually responsible for the effects seen to the dependent variable (Jedlitschka et al., 2008). Experiments were performed using balanced training sets, which included the same number of successful and unsuccessful issues to avoid the effects of a variable training set. In some experiments considering measures of time of more than 2000 days, few data were used to perform experiments, so, in these cases, results may have been influenced by the amount of training data; however, more than 6000 experiments were performed to observe the variation of issue success predictions over time. In addition, data preprocessing was a validated process, so threats related to the effect of manipulations on results were mitigated.

External validity refers to the degree to which the findings of the study can be generalized to other participant populations or settings (Jedlitschka et al., 2008). In this work, data from four public Jira ITSs that store a considerable number of software projects were selected as data source. The selected Jira ITSs are some of the most used ITSs due to the kind of projects that store, which include the development of software products that are used by a lot of people that are dedicated to software engineering. These software products include software development tools, plugins, libraries, frameworks, programming languages, IDEs, and collaboration tools, and many of them are Apache, Spring and Atlassian products; thus, this study must be of the interest of a lot of people dedicated to software development. The data that were extracted from the ITSs include issues and comments from 588 software projects (representing more than 50% of the total projects of the selected ITSs), so the studied data are a representative part of the projects that are recorded in public Jira ITSs. In addition, more than 6000 experiments were performed to study the prediction of issue success considering different variables (algorithms, features, weighting schemes, issue types, periods of time) with the aim of considering most of the possible cases and scenarios.

Conclusion validity refers to whether the conclusions reached in a study are correct (Jedlitschka et al., 2008). In this study, conclusions of results were stated considering the limitations and context of this work.

Reliability is concerned with the extent to which the data and the analysis are dependent on specific researchers (Runeson et al., 2012). The studied data were extracted directly from four Jira ITSs, and they were not modified by any researcher. The data analysis was automatic, so did not depend on specific researchers. Most of the experiments were executed several times to provide reliability to the study. The data extraction, data preprocessing, and the experiment execution activities were validated.

7. Conclusions and future work

In this work, the usefulness of descriptions of issues and comments for predicting issue success was studied. More than 6000 experiments were performed considering seven machine learning classifiers, 80 measures of time and three types of issues (bugs, improvements and new features).

Results showed that descriptions of issues and comments can be used for predicting issue success with good levels of accuracy and precision. Some words that relate to software development were particularly useful for predicting issue success.

The prediction of issue success can reduce costs in software development and improve software quality because people can identify the issues that will not be successfully addressed. This can be useful for performing actions to avoid issue failure and reduce development time.

More research is needed on predicting issue success, and data form other repositories and tools must be studied. Other aspects of people interaction and human factors must be considered as features for performing predictions in software development.

CRediT authorship contribution statement

Sandra L. Ramírez-Mora: Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Writing - original draft, Visualization. **Hanna Oktaba:** Conceptualization, Methodology, Validation, Writing - review & editing, Supervision. **Helena Gómez-Adorno:** Conceptualization, Methodology, Software, Validation, Writing - review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Machine learning classifiers

Multinomial Naïve Bayes (MNB)

Naïve Bayes is a probabilistic classifier that uses the Bayes' theorem, which expresses the probability of an aleatory event knowing conditions that might be related to the event. Naive Bayes is widely used in machine learning due to its efficiency and its ability to combine evidence from a large number of features (Basu et al., 2003; Mitchell, 1997). Naive Bayes assumes that the value of a particular feature is independent of the value of any other feature, given the class variable (Manning and Schütze, 1999). Naïve Bayes classifier can be extremely fast compared to more sophisticated methods and they require a small amount of training data to estimate the necessary parameters (Pedregosa et al., 2011; Zhang, 2004). The scikit-learn library implements the Multinomial Naïve Bayes (MNB) classifier, which is a Naive Bayes algorithm variant that is recommended for text classification. The parameters of MNB that were used in this work are the following.

MultinomialNB (alpha=1.0, class_prior=None, fit_prior=True)

Logistic Regression (LR)

Logistic Regression (LR) is also known as logit regression, maximum-entropy classification or the log-linear classifier. LR, despite its name, is a linear model for classification rather than regression in which the probabilities describing the possible outcomes of a single trial are modeled using a logistic function (Pedregosa et al., 2011). The scikit-learn library implements the LR classifier. The parameters of LR that were used in this work are the following.

LogisticRegression (C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False)

Support Vector Classifier (SVC)

A Support Vector Machine (SVM) is a non-probabilistic binary linear classifier (Garreta and Moncecchi, 2013). It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient (Seal, 1967). SVM is effective in high dimensional spaces; still effective in cases where the number of dimensions is greater than the number of samples (Pedregosa et al., 2011). The scikit-learn library implements the Support Vector Classifier (SVC), which is an SVM classifier. The parameters of SVC that were used in this work are the following.

SVC (C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='rbf', max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001, verbose=False)

Decision Tree Classifier (DTC)

Decision Tree is a tree-like structure that is used for classification and regression. Each internal node of a decision tree denotes a test on an attribute, each branch represents an outcome of the test and each leaf node represents a class label. The scikit-learn library implements the Decision Tree Classifier (DTC), which is capable of performing multi-class classification on a dataset (Pedregosa et al., 2011). The parameters of DTC that were used in this work are the following.

DecisionTreeClassifier (class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

Multi-Layer Perceptron Classifier (MLPC)

Artificial Neural Networks (ANN) are computing systems that consist of a set of unities called neurons that are connected to perform classifications or predictions. The MLP Classifier (MLPC) is an ANN model that implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation (Pedregosa et al., 2011). MLPC is implemented in the scikit-learn library and the parameters of MLPC that were used in this work are the following.

MLPClassifier (activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(5, 2), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

Ensemble Methods

Ensemble methods combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability/robustness over a single estimator (Pedregosa et al., 2011). Two families of ensemble methods are usually distinguished.

- **Averaging methods**, in which the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is

Table B.8
Confusion matrix.

		Predicted values	
		Unsuccessful issues	Unsuccessful issues
Actual values	Successful issues	True Positives (TP)	False Negatives (FN)
	Unsuccessful issues	False Positives (FP)	True Negatives (TN)

Table C.9
Accuracy results of predicting successful and unsuccessful issues.

	MNB	LR	SVC	DTC	MLPC	RFC	GBC
BUGS							
Min.	0.72	0.66	0.38	0.66	0.69	0.63	0.69
Max.	0.93	1	0.71	0.88	0.9	0.9	0.98
Mean	0.81	0.82	0.48	0.76	0.81	0.75	0.81
Variance	0	0	0	0	0	0	0
Standard deviation	0.04	0.05	0.04	0.05	0.04	0.05	0.05
IMPROVEMENTS							
Min.	0.33	0.25	0.25	0.25	0.31	0.25	0.42
Max.	0.83	0.81	0.68	0.78	0.76	0.76	0.79
Mean	0.64	0.64	0.46	0.58	0.62	0.58	0.65
Variance	0.01	0.01	0	0.01	0.01	0.01	0.01
Standard deviation	0.09	0.1	0.06	0.08	0.1	0.08	0.07
NEW FEATURES							
Min.	0.33	0.33	0.33	0.22	0.33	0.22	0.33
Max.	1	0.96	0.67	0.85	0.89	0.89	0.9
Mean	0.7	0.69	0.47	0.6	0.68	0.61	0.68
Variance	0.01	0.01	0	0.01	0.01	0.01	0.01
Standard deviation	0.1	0.09	0.05	0.09	0.08	0.11	0.09

Table C.10
Precision results of predicting successful and unsuccessful issues.

	MNB	LR	SVC	DTC	MLPC	RFC	GBC
SUCCESSFUL BUGS							
Min.	0.61	0.59	0	0.51	0.59	0.46	0.49
Max.	1	1	0.57	0.98	1	1	1
Mean	0.77	0.74	0.34	0.72	0.77	0.63	0.69
Variance	0.01	0.01	0.02	0.01	0.01	0.02	0.01
Standard deviation	0.09	0.1	0.14	0.11	0.1	0.12	0.11
UNSUCCESSFUL BUGS							
Min.	0.65	0.71	0.39	0.5	0.64	0.46	0.49
Max.	1	0.95	0.57	0.88	0.91	1	1
Mean	0.82	0.84	0.45	0.76	0.81	0.63	0.69
Variance	0	0	0	0	0	0.02	0.01
Standard deviation	0.06	0.05	0.04	0.07	0.05	0.12	0.11
SUCCESSFUL IMPROVEMENTS							
Min.	0.08	0.08	–	0.12	0.11	0.05	0
Max.	0.86	0.61	–	0.59	0.57	0.51	0.67
Mean	0.36	0.31	–	0.27	0.27	0.24	0.3
Variance	0.02	0.01	–	0.01	0.01	0.01	0.01
Standard deviation	0.14	0.11	–	0.1	0.1	0.1	0.12
UNSUCCESSFUL IMPROVEMENTS							
Min.	0.65	0.65	–	0.61	0.54	0.05	0
Max.	1	1	–	1	1	0.51	0.67
Mean	0.86	0.86	–	0.84	0.85	0.24	0.3
Variance	0	0	–	0.01	0	0.01	0.01
Standard deviation	0.06	0.06	–	0.07	0.07	0.1	0.12
SUCCESSFUL NEW FEATURES							
Min.	0.03	0.03	–	0	0	0	0
Max.	0.62	0.63	–	0.58	0.59	0.54	0.6
Mean	0.26	0.26	–	0.22	0.24	0.22	0.25
Variance	0.02	0.02	–	0.02	0.02	0.02	0.02
Standard deviation	0.15	0.15	–	0.14	0.15	0.12	0.14
UNSUCCESSFUL NEW FEATURES							
Min.	0.68	0.7	–	0.62	0.69	0	0
Max.	1	1	–	1	1	0.54	0.6
Mean	0.89	0.88	–	0.84	0.86	0.22	0.25
Variance	0	0	–	0.01	0	0.02	0.02
Standard deviation	0.07	0.07	–	0.08	0.07	0.12	0.14

Table C.11
Recall results of predicting successful and unsuccessful issues.

	MNB	LR	SVC	DTC	MLPC	RFC	GBC
SUCCESSFUL BUGS							
Min.	0.53	0.66	0	0.46	0.55	0.58	0.64
Max.	1	0.95	1	0.87	0.92	0.97	0.96
Mean	0.71	0.76	0.86	0.62	0.7	0.7	0.78
Variance	0.01	0.01	0.12	0.01	0.01	0.01	0.01
Standard deviation	0.11	0.08	0.34	0.07	0.08	0.07	0.08
UNSUCCESSFUL BUGS							
Min.	0.76	0.74	0	0.74	0.73	0.61	0.53
Max.	1	1	1	0.97	1	1	1
Mean	0.86	0.83	0.14	0.84	0.86	0.72	0.77
Variance	0	0	0.12	0	0	0.01	0.01
Standard deviation	0.05	0.06	0.34	0.06	0.05	0.08	0.08
SUCCESSFUL IMPROVEMENTS							
Min.	0.2	0.25	–	0.17	0.25	0.13	0
Max.	1	1	–	1	1	1	1
Mean	0.52	0.61	–	0.53	0.63	0.58	0.62
Variance	0.02	0.02	–	0.02	0.01	0.01	0.02
Standard deviation	0.15	0.12	–	0.13	0.11	0.11	0.13
UNSUCCESSFUL IMPROVEMENTS							
Min.	0.4	0.33	–	0.23	0.31	0.22	0.42
Max.	0.98	0.89	–	0.82	0.65	0.71	0.94
Mean	0.73	0.65	–	0.62	0.55	0.53	0.63
Variance	0.01	0.01	–	0.01	0	0.01	0.01
Standard deviation	0.12	0.1	–	0.09	0.06	0.08	0.09
SUCCESSFUL NEW FEATURES							
Min.	0.25	0.25	–	0	0	0	0
Max.	1	1	–	1	1	1	1
Mean	0.69	0.67	–	0.47	0.58	0.56	0.61
Variance	0.02	0.02	–	0.02	0.03	0.02	0.04
Standard deviation	0.14	0.14	–	0.16	0.17	0.15	0.2
UNSUCCESSFUL NEW FEATURES							
Min.	0.07	0.13	–	0.18	0.25	0.17	0.19
Max.	0.72	0.7	–	0.92	0.74	0.69	0.84
Mean	0.53	0.54	–	0.6	0.55	0.52	0.59
Variance	0.03	0.02	–	0.02	0.01	0.01	0.01
Standard deviation	0.16	0.15	–	0.13	0.11	0.09	0.11

usually better than any of the single base estimator because its variance is reduced (Pedregosa et al., 2011). The scikit-learn library includes an averaging algorithm based on randomized decision trees: Random Forest Classifier (**RFC**) (Pedregosa et al., 2011). RFC perturb-and-combine techniques (Breiman, 1998) specifically designed for trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers (Pedregosa et al., 2011). Random Forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting (Breiman, 2001; Seal, 1967). The parameters of RFC that were used in this work are the following.

RandomForestClassifier (bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start =False)

- **Boosting methods**, in which base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble (Pedregosa et al., 2011). Gradient boosting (GB) is a machine learning technique to perform classification and regression tasks using prediction models, typically decision trees. The classifier builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions (Seal, 1967). Gradient Boosting Classifier (**GBC**) supports both binary and multi-class classification and is part of the scikit-learn library (Pedregosa et al., 2011). The parameters of GBC that were used in this work are the following.

GradientBoostingClassifier (criterion='friedman_mse', init= None, learning_rate=0.1, loss='deviance', max_depth=3, max_features= None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf =1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)

Appendix B. Measures for evaluating machine learning algorithms

The following measures are based on variables in Table B.8.

Precision. The ratio of correctly predicted observations in a class to the total predicted observations in such class; (B.1) and (B.2) were used to calculate the precision for predicting successful and unsuccessful issues respectively.

$$Precision_1 = \frac{TP}{TP + FP} \tag{B.1}$$

$$Precision_2 = \frac{TN}{TN + FN} \tag{B.2}$$

Table C.12
F1-score results of predicting successful and unsuccessful issues.

	MNB	LR	SVC	DTC	MLPC	RFC	GBC
SUCCESSFUL BUGS							
Min.	0.6	0.65	0	0.53	0.62	0.54	0.59
Max.	1	0.96	0.72	0.92	0.94	0.93	0.96
Mean	0.73	0.75	0.48	0.66	0.73	0.66	0.73
Variance	0.01	0.01	0.04	0.01	0.01	0.01	0.01
Standard deviation	0.1	0.08	0.2	0.08	0.08	0.09	0.09
UNSUCCESSFUL BUGS							
Min.	0.73	0.73	0.57	0.6	0.69	0.61	0.64
Max.	1	0.96	0.72	0.91	0.94	0.91	0.95
Mean	0.84	0.83	0.62	0.79	0.83	0.75	0.8
Variance	0	0	0	0	0	0	0
Standard deviation	0.05	0.04	0.04	0.05	0.04	0.05	0.06
SUCCESSFUL IMPROVEMENTS							
Min.	0.13	0.13	–	0.15	0.18	0.07	0
Max.	0.63	0.64	–	0.57	0.63	0.6	0.67
Mean	0.4	0.4	–	0.35	0.37	0.33	0.39
Variance	0.01	0.01	–	0.01	0.01	0.01	0.01
Standard deviation	0.1	0.1	–	0.1	0.11	0.1	0.11
UNSUCCESSFUL IMPROVEMENTS							
Min.	0.55	0.45	–	0.38	0.44	0.3	0.57
Max.	0.93	0.89	–	0.83	0.76	0.77	0.91
Mean	0.78	0.73	–	0.71	0.67	0.64	0.72
Variance	0.01	0.01	–	0	0	0.01	0
Standard deviation	0.07	0.07	–	0.07	0.05	0.08	0.06
SUCCESSFUL NEW FEATURES							
Min.	0.05	0.05	–	0	0	0	0
Max.	0.64	0.65	–	0.56	0.64	0.58	0.64
Mean	0.35	0.35	–	0.28	0.32	0.3	0.33
Variance	0.02	0.02	–	0.02	0.02	0.02	0.02
Standard deviation	0.14	0.14	–	0.13	0.15	0.13	0.14
UNSUCCESSFUL NEW FEATURES							
Min.	0.13	0.22	–	0.29	0.38	0.28	0.33
Max.	0.79	0.78	–	0.86	0.79	0.77	0.9
Mean	0.64	0.66	–	0.69	0.66	0.64	0.69
Variance	0.02	0.02	–	0.01	0.01	0.01	0.01
Standard deviation	0.15	0.13	–	0.1	0.08	0.08	0.09

Accuracy. The ratio of correctly predicted observations to the total observations.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{B.3}$$

Recall. The ratio of correctly predicted observations in a class to the all observations in such actual class; (B.4) and (B.5) were used to calculate the recall for predicting successful and unsuccessful issues respectively.

$$Recall_1 = \frac{TP}{TP + FN} \tag{B.4}$$

$$Recall_2 = \frac{TN}{TN + FP} \tag{B.5}$$

F1 score. The weighted average of precision and recall.

$$F1\_score = \frac{2 * (Recall * Precision)}{Recall + Precision} \tag{B.6}$$

Appendix C. Descriptive statistics of results

Descriptive statistics are summarized in Tables C.9–C.12.

References

Akkermans, H., van Helden, K., 2002. Vicious and virtuous cycles in ERP implementation: a case study of interrelations between critical success factors. Eur. J. Inf. Syst. 11 (1), 35–46. http://dx.doi.org/10.1057/palgrave.ejis.3000418.

Basu, A., Watters, C., Shepherd, M., 2003. Support vector machines for text categorization. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 4 - Volume 4. HICSS '03, IEEE Computer Society, Washington, DC, USA, p. 103.3. http://dx.doi.org/10.1109/HICSS.2003.1174243. URL http://dl.acm.org/citation.cfm?id=820751.821531.

Binkley, D., Feild, H., Lawrie, D., Pighin, M., 2009. Increasing diversity: Natural language measures for software fault prediction. J. Syst. Softw. 82 (11), 1793–1803, SI: TAIC PART 2007 and MUTATION 2007. http://dx.doi.org/10.1016/j.jss.2009.06.036. URL http://www.sciencedirect.com/science/article/pii/S0164121209001332.

Breiman, L., 1998. Arcing classifiers. Ann. Stat. 26, 801–823.

Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32. http://dx.doi.org/10.1023/A:1010933404324.

Butler, T., 2003. An institutional perspective on developing and implementing intranet- and internet-based information systems. Inf. Syst. J. 13, 209–231. http://dx.doi.org/10.1046/j.1365-2575.2003.00151.x.

Butler, T., Fitzgerald, B., 2001. The relationship between user participation and the management of change surrounding the development of information systems: A european perspective. J. End User Comput. 12–25 (1), 12. http://dx.doi.org/10.4018/joeuc.2001010102.

Catal, C., 2011. Review: Software fault prediction: A literature review and current trends. Expert Syst. Appl. 38 (4), 4626–4636. http://dx.doi.org/10.1016/j.eswa.2010.10.024.

Choetkiertikul, M., Dam, H.K., Tran, T., Ghose, A., Grundy, J., 2018. Predicting delivery capability in iterative software development. IEEE Trans. Softw. Eng. 44 (6), 551–573. http://dx.doi.org/10.1109/TSE.2017.2693989.

Destefanis, G., Ortu, M., Counsell, S., Swift, S., Marchesi, M., Tonelli, R., 2016. Software development: do good manners matter?. PeerJ Comput. Sci. 2, e73. http://dx.doi.org/10.7717/peerj-cs.73.

Di Sorbo, A., Spillner, J., Canfora, G., Panichella, S., 2019. "won't we fix this issue?" Qualitative characterization and automated identification of wontfix issues on github. CoRR abs/1904.02414, arXiv:1904.02414.

Duc Anh, N., Cruzes, D.S., Conradi, R., Ayala, C., 2011. Empirical validation of human factors in predicting issue lead time in open source projects. In:

Proceedings of the 7th International Conference on Predictive Models in Software Engineering. Promise '11, ACM, New York, NY, USA, pp. 13:1–13:10. http://dx.doi.org/10.1145/2020390.2020403.

Fagerholm, F., Ikonen, M., Kettunen, P., Münch, J., Roto, V., Abrahamsson, P., 2015. Performance alignment work: How software developers experience the continuous adaptation of team performance in lean and agile environments. Inf. Softw. Technol. 64, 132–147. http://dx.doi.org/10.1016/j.infsof.2015.01.010, URL http://www.sciencedirect.com/science/article/pii/S0950584915000269.

Fronza, I., Sillitti, A., Succi, G., Terho, M., Vlasenko, J., 2013. Failure prediction based on log files using random indexing and support vector machines. J. Syst. Softw. 86 (1), 2–11. http://dx.doi.org/10.1016/j.jss.2012.06.025, URL http://www.sciencedirect.com/science/article/pii/S0164121212001732.

Garousi, V., Tarhan, A., Pfahl, D., Coşkunçay, A., Demirörs, O., 2019. Correlation of critical success factors with success of software projects: an empirical investigation. Softw. Qual. J. 27 (1), 429–493. http://dx.doi.org/10.1007/s11219-018-9419-5.

Garreta, R., Moncecchi, G., 2013. Learning Scikit-learn: Machine Learning in Python. Packt Publishing.

Guo, P.J., Zimmermann, T., Nagappan, N., Murphy, B., 2010. Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1. ICSE '10, Association for Computing Machinery, New York, NY, USA, pp. 495–504. http://dx.doi.org/10.1145/1806799.1806871, URL https://doi-org.pbidi.unam.mx:2443/10.1145/1806799.1806871.

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., 2012. A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. 38 (6), 1276–1304. http://dx.doi.org/10.1109/TSE.2011.103.

Hartwick, J., Barki, H., 2001. Communication as a dimension of user participation. IEEE Trans. Prof. Commun. 44 (1), 21–36. http://dx.doi.org/10.1109/47.911130.

Hoegl, M., Gemuenden, H.G., 2001. Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. Informs 12, 435–449. http://dx.doi.org/10.1287/orsc.12.4.435.10635.

Jakobson, R., 1963. Essais de linguistique générale. In: Points (Paris), Les Editions de Minuit, URL https://books.google.com.mx/books?id=CsOPvgEACAAJ.

Jedlitschka, A., Ciolkowski, M., Pfahl, D., 2008. Reporting Experiments in Software Engineering. Springer, London, pp. 201–228. http://dx.doi.org/10.1007/978-1-84800-044-5_8.

Jones, K.S., 1972. A statistical interpretation of term specificity and its application in retrieval. J. Doc. 28, 11–21.

Lindsjørn, Y., Sjøberg, D.I., Dingsøyr, T., Bergersen, G.R., Dybå, T., 2016. Teamwork quality and project success in software development: A survey of agile development teams. J. Syst. Softw. 122, 274–286. http://dx.doi.org/10.1016/j.jss.2016.09.028, URL http://www.sciencedirect.com/science/article/pii/S016412121630187X.

Littlejohn, S., Foss, K., 2009. Encyclopedia of Communication Theory, vol. 1, SAGE Publications, URL https://books.google.com.mx/books?id=S8Kf0N0XALIC.

Manning, C.D., Schütze, H., 1999. Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA, USA.

McLeod, L., MacDonell, S.G., 2011. Factors that affect software systems development project outcomes: A survey of research. ACM Comput. Surv. 43 (4), 24:1–24:56. http://dx.doi.org/10.1145/1978802.1978803, URL http://doi.acm.org/10.1145/1978802.1978803.

Melo, C., Cruzes, D.S., Kon, F., Conradi, R., 2013. Interpretative case studies on agile team productivity and management. Inf. Softw. Technol. 55 (2), 412–427, Special Section: Component-Based Software Engineering (CBSE), 2011. http://dx.doi.org/10.1016/j.infsof.2012.09.004. URL http://www.sciencedirect.com/science/article/pii/S0950584912001875.

Menzies, T., Greenwald, J., Frank, A., 2007. Data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng. 33 (1), 2–13. http://dx.doi.org/10.1109/TSE.2007.10.

Mitchell, T.M., 1997. Machine Learning, first ed. McGraw-Hill, Inc., New York, NY, USA.

Murgia, A., Concas, G., Tonelli, R., Ortu, M., Demeyer, S., Marchesi, M., 2014. On the influence of maintenance activity types on the issue resolution time. In: Proceedings of the 10th International Conference on Predictive Models in Software Engineering. PROMISE '14, Association for Computing Machinery, New York, NY, USA, pp. 12–21. http://dx.doi.org/10.1145/2639490.2639506.

Nasir, M., Sahibuddin, S., 2011. Critical success factors for software projects: A comparative study. Sci. Res. Essays 6, 2174–2186. http://dx.doi.org/10.5897/SRE10.1171.

Ortu, M., Adams, B., Destefanis, G., Tourani, P., Marchesi, M., Tonelli, R., 2015. Are bullies more productive? empirical study of affectiveness vs. issue fixing time. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. pp. 303–313. http://dx.doi.org/10.1109/MSR.2015.35.

Pearson, K., 1895. Note on regression and inheritance in the case of two parents. Proceedings of the Royal Society of London 58, 240–242, https://www.bibsonomy.org/bibtex/25f8cf240981fc543a2192aa475a0f88d/thoni.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Rajaraman, A., Ullman, J.D., 2011. Mining of Massive Datasets. Cambridge University Press, New York, NY, USA.

Ramírez-Mora, S.L., Oktaba, H., 2017. Productivity in agile software development: A systematic mapping study. In: 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT). pp. 44–53. http://dx.doi.org/10.1109/CONISOFT.2017.00013.

Ramírez-Mora, S.L., Oktaba, H., Gómez-Adorno, H., 2020. Issues, comments and projects from four popular Issue Tracking Systems. "Mendeley Data", http://dx.doi.org/10.17632/pk3wv93s3m.1.

Runeson, P., Host, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples, first ed. Wiley Publishing.

Sawyer, S., Guinan, P.J., 1998. Software development: Processes and performance. IBM Syst. J. 37 (4), 552–569. http://dx.doi.org/10.1147/sj.374.0552.

Schermerhorn, J., Hunt, J., Osborn, R., 2002. Organizational Behavior. John Wiley & Sons.

Seal, H.L., 1967. Studies in the history of probability and statistics. XV the historical development of the Gauss linear model. Biometrika 54 (1–2), 1–24. http://dx.doi.org/10.1093/biomet/54.1-2.1.

Shippey, T., Bowes, D., Hall, T., 2019. Automatically identifying code features for software defect prediction: using AST N-grams. Inf. Softw. Technol. 106, 142–160. http://dx.doi.org/10.1016/j.infsof.2018.10.001, URL http://www.sciencedirect.com/science/article/pii/S0950584918302052.

Somers, T.M., Nelson, K., 2001. The impact of critical success factors across the stages of enterprise resource planning implementations. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences. p. 10. http://dx.doi.org/10.1109/HICSS.2001.927129.

Stigler, S.M., 1989. Francis galton's account of the invention of correlation. Statistical Science 4 (2), 73–79, http://www.jstor.org/stable/2245329.

Suma, V., Pushphavathi, T.P., Ramaswamy, V., 2014. An approach to predict software project success based on random forest classifier. In: Satapathy, S.C., Avadhani, P.S., Udgata, S.K., Lakshminarayana, S. (Eds.), ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India- Vol II. Springer International Publishing, Cham, pp. 329–336. http://dx.doi.org/10.1007/978-3-319-03095-1_36.

Valdivia-Garcia, H., Shihab, E., Nagappan, M., 2018. Characterizing and predicting blocking bugs in open source projects. J. Syst. Softw. 143, 44–58. http://dx.doi.org/10.1016/j.jss.2018.03.053, URL http://www.sciencedirect.com/science/article/pii/S0164121218300530.

Wu, H.C., Luk, R.W.P., Wong, K.F., Kwok, K.L., 2008. Interpreting TF-IDF term weights as making relevance decisions. ACM Trans. Inf. Syst. 26 (3), 13:1–13:37. http://dx.doi.org/10.1145/1361684.1361686.

Zhang, H., 2004. The optimality of naive Bayes. In: Barr, V., Markov, Z. (Eds.), Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004). AAAI Press, pp. 562–567.

Sandra L. Ramírez-Mora received her Bachelor and Master degrees in Computer Engineering from the National Autonomous University of Mexico, Mexico. She has been involved in several agile projects as analyst, developer, designer and quality assurance. She is currently a Ph.D. student at the National Autonomous University of Mexico. Her research focuses on human factors in software engineering, agile methods, productivity in software development and artificial intelligence techniques in software engineering.

Hanna Oktaba holds a Ph.D. in Computer Science from the University of Warsaw, Poland. From 1983 she is professor of the National Autonomous University of Mexico in the area of Software Engineering. She was in charge of the project to create the process model for small software organizations MoProSoft. Since 2006, she participated in the creation of the ISO/IEC 29110 standard for Very Small Entities (VSEs). In 2011, she presented the proposal of KUALI-BEH: Software Project Common Concepts, which was integrated into the ESSENCE 1.0 proposal published as an OMG standard in November 2014.

Helena Gómez-Adorno is an associate researcher at Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, UNAM. She finished her Ph. D. in Computer Science at the Centro de Investigación en Computación, IPN. She completed a postdoctoral stay at the Linguistic Engineering Group of the Engineering Institute of the UNAM. Her research interests are in the field of natural language processing. She has worked on question answering systems, semantic similarity, authorship attribution, author profiling, and text classification problems. She is a current member of the National System of Researchers of CONACYT Level 1.