TOR VERGATA
UNIVERSITY OF ROME

SCHOOL OF ENGINEERING

MASTER DEGREE ON
COMPUTER SCIENCE ENGINEERING

A.A. 2023/2024

Thesis

Predicting bug-inducing tickets: the impact of temporal proximity

RELATORE

Chiarissimo Prof. Davide Falessi

CANDIDATO

Daniele La Prova
0320429

CORRELATORI

Emanuele Gentili @ MBDA Italia SPA

"Make the money, don't let the money make you" - Macklemore.

"When you are a student, the fundamental goal is to grow, to learn, to become better [. . .] In life after college, the goal changes a little, and success hinges on how effectively you're able to add value to others." - Grant Sanderson (3Blue1Brown).

"It's not a bug, it's a feature!" - Anonymous.

It is easier to search where the light is, but someone has to venture the dark first. This work is dedicated to those who dare to get out of their comfort zone following their purpose despite having fear of the unknown like all us do.

Contents

1 Acknowledgment

This section is in italian since all the people mentioned here are italian and I want to make sure they understand my gratitude.

Questa è la parte più delicata, ma è anche la mia preferita.

Alla mia famiglia tutta, in particolare a Mamma, Papà e mio fratello Federico, perché tra tutte le cose loro sono stati sempre in prima linea a supportarmi e sopportarmi dietro le quinte di questo percorso. A Nonno Vincenzo e Nonna Natalina, A Nonno Ivo e Nonna Carmela, a Zio Gianni e Zia Pina, Zia Lina, zia Elvira e Zio Nicola, Zia Albina e Zio Lauso, e ai miei cugini Alessandro ed Emanuele. Grazie di cuore.

Ai miei Pistoleri, ovvero Alessandro, Giacomo, Marcello, Matteo, Riccardo, Claudia, Farida e Syria. Con i ragazzi ne abbiamo passate tante, e con l'aggiunta delle ragazze ne stiamo passando altrettante, vivendo dinamiche di tutti i colori che hanno un che di fraterno, e di cui sono genuinamente grato di farne parte.

Ad Alessandro G., Andrea Lin, Andrea P. M., Avril, Elena, Giulia, Luca F., Matteo C., Michele, Riccardo S., e a Tiziano. Chi prima e chi dopo, vi ho conosciuto tutti nei corridoi dell'università. Ci hanno avvicinato le necessità, ma ci ha unito un'amicizia che spero sopravviva al tempo e alle distanze. Grazie per essere rimasti. Tra questi, vorrei spendere altre due parole per Michele. Neanche sospetti la solidità del pilastro che la nostra amicizia ha eretto nel mio palazzo mentale. Per me è un dono prezioso di cui sono grato di avere a disposizione nei momenti di maggiore difficoltà. Grazie mille bro.

Vorrei ringraziare tutti i colleghi Tutor, professori e le matricole con cui ho interagito durante le attività di Tutoraggio. Non avete idea di quanto io abbia imparato da queste esperienze, e di quanto sono grato delle nuove amicizie che cosi ho stretto. In particolare ringrazio Raffaele, Giulia, Eleonora e Benedetta

A tutti i colleghi e amici che ho incontrato durante il percorso, che sia per preparare un esame insieme o solo per una chiacchera davanti a un caffé. Ho imparato da tutti voi, e vi ringrazio per questo.

Ai miei docenti, a cui devo innumerevoli occasioni di crescita tecnica e personale. In particolare, al Professore Davide Falessi, che fin dal primo momento ha seguito lo sviluppo della Tesi con vivo interesse e totale disponibilità, valorizzando le mie opinioni in materia. Professore, spero che il nostro lavoro contribuisca a portare un po' di luce dove abbiamo avuto difficoltà a trovarne.

Un ringraziamento sentito lo dedico all'Ingegnere Emanuele Gentili, che ha fornito un prezioso contributo alla stesura di questo lavoro senza il quale non avrebbe lo stesso valore.

A tutti coloro che non ho nominato ma che hanno indubbiamente fatto parte della mia vita. Grazie. Spero tu abbia un bel ricordo di me come io lo ho di te.

2 Introduction

Modern society heavily relies on software, permeating all aspects of our lives and lifting people from the burden of complex but sometimes critical tasks. For this reason, it is of upmost importance that we as users can rely on the software we depend on to be effective. When software fails, the costs can be immense. According to Mugu et al. [73], 3 out of the 5 most recent dire outages are related to bugs (Figure 1), with the most recent being the infamous 2024 CrowdStrike Incident, where the company pushed to production a buggy software update unnoticed by the (possibly insufficient) QA checks. The faulty patch caused an outage of many Windows systems , resulting in

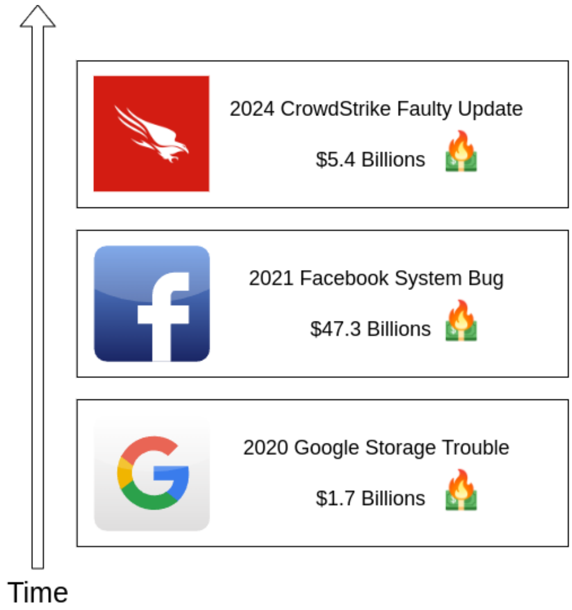


Figure 1: Most recent dire software outages resulting in heavy losses.

disruption of services in several critical sectors, including healthcare, transportation, and finance, all around the globe [74], with an initial estimated cost in damages estimated from 4 to 6 billion dollars [39]. The take home message from this example is that the later a bug is found, the more expensive its aftermaths are, so finding and fixing bugs in time is of the essence.

Software engineering comes into play here as the discipline that deals with the cost efficiency, good quality and timely delivery of software systems using quantitative and measurable approaches [40]. When developing a project using SE principles, it is not uncommon that developers organize requirements in tickets since they are a good way to keep track of the work [85].

Since bugs can have a significant impact on time, budget and safety, much effort has been spent on bug prediction. The main purpose of bug prediction is to minimize the testing effort. This is achieved by focusing testing on specific software artifacts, such as classes, methods, commits, or lines of code predicted to be buggy. Consequently, significant progress has been made in developing prediction models at the class, method, commit, and line levels [57, 107, 107, 68, 89, 35, 91, 23, 72, 86, 25, 76]. However, these predicted entities already contain bugs.

Change impact analysis is a software engineering research area focusing on assessing the consequences of modifications to software systems so that teams can minimize unintended consequences, optimize their development efforts, and establish maintenance and testing strategies [64, 11, 5, 37, 6].

Requirements quality refers to the degree to which software requirements are well-defined, unambiguous, complete, consistent, and testable [9, 99]. High-quality requirements are essential for guiding development teams and ensuring the final product aligns with stakeholders' needs [97]. Berry and Lawrence [9] emphasize that clear and precise requirements minimize misunderstandings during development, leading to more efficient project execution, and several studies [3, 51, 10] report on the disruptive effects that ambiguity, inconsistency, incompleteness or, more generally, "requirements smells" [38] can have on SW project success [56].

Models prediction accuracy is impacted by time [12, 88, 13, 75], and it is reasonable to assume that the closer we get to the prediction instant, the more information we gain, and the more precise the prediction becomes. So, the concept of temporal proximity in prediction comes to play a significative role, as prediction accuracy typically declines over longer time horizons due to the error accumulation of long-term predictors [12, 75], especially when applied to intrinsically stochastic processes, like weather forecasting [13, 70], financial stock market [69], or epidemic modelling [59].

With the idea that prevention is better than cure, our aim is to propose and evaluate a first approach for ticket-level prediction (TLP); the approach predicts which tickets, when implemented, will lead to bug injection.

We consider three temporal points to characterize the lifecycle of a ticket: created, assigned, and implemented. We investigate how temporal proximity impacts TLP in terms of the accuracy and power of the predictive features. Specifically, we conjecture that: 1) TLP accuracy improves as the ticket moves closer to implementation (i.e., the bug moves closer to its injection) due to the increasing reliability of predictive features over time, and 2) the power of predictive features changes over time.

As TLP features, we propose and measure 62 features coming from commit-level and class-level defect prediction, requirements quality, NLP, and the broader software engineering domain.

Our TLP evaluation considers balancing, feature selection, and many machine-learning bug prediction classifiers on about 11,000 tickets related to two open-source projects from the Apache ecosystem. As TLP accuracy metrics we use Precision, Recall, F1, AUC, Kappa and GMean. As TLP power metrics we use the info gain ratio and backward search feature selection.

Our results show that SlidingWindow is overall more accurate than 80-20 in AUC. The AUC of SlidingWindow of Closed is about 1. Therefore, Practitioners should use a sliding window approach given the concept drift in TLP as observed in JIT [72] and Researchers should focus on the sliding window approach when assessing the power of prediction features. Results show that, as expected, the TLP accuracy increases when proximity increases. The difference in TLP accuracy across temporal points is statistically significant in all 24 cases. Regarding the results of the prediction power of feature families, our study shows that the power of feature families changes according to the feature family, the proximity and their interaction. Therefore, no single feature family is more powerful than another independently of the proximity points. Therefore, Practitioners should feed TLP models with specific features according to the proximity point. In particular, In closed, the feature families different from JIT decrease their selection proportion but are still selected, and their IGR does not change. Therefore, for TLP in Closed, JIT is by far the most important feature family, although feature families different from JIT are still useful for prediction.

The remainder of this paper is structured as it follows. Section 4 describes the empirical study design. Section 5 reports and discusses the results of the empirical investigation. Section 6 discusses the threats to the study validity. Finally, section 8 concludes the paper and outlines directions for future work.

3 TLP Features

This section describes the features we use to perform TLP, which have been summarized in Table 1. The features are organized into families. To find features, we started with main studies on defect prediction on all granularities, i.e., classes, methods, tickets and lines [25] [68] [76]. Afterward, we increased the number of papers analyzed by snowballing [102]. Finally, we complemented this set of papers by looking at papers about quality factors in the description of requirements. From this set of tickets, we derived 62 features, as detailed in the following.

Table 1: TLP features. There are a total of 47 features, of which: 4 belong to Code family, 2 to Developer, 6 to External Temperature, 10 to Internal Temperature, 22 to Intrinsic, 3 to R2R.

Name	CodeName	Family	Reference	Availability
Number of different Languages	code_size-number_of_languages	Code	[61]	Open
Number of Files	code_size-number_of_files	Code	[104]	Open
Project Code Quality	code_quality-Smells_count	Code	[16] [26]	Open
Total LOCs	code_size-total_LOCs	Code	[104]	Open
Assigned Developer's ANFIC	assignee-ANFIC	Developer	[71]	Assigned
Assigned Developer's Familiarity	assignee-Familiarity	Developer	[96]	Assigned
Churn of Commits while in progress	commits_while_in_progress-Churn	External Temperature	[78] [29]	Assigned
Number of Commits while in progress	commits_while_in_progress-Count	External Temperature	[78]	Assigned
Latest commit churn	latest_commit-Churn	External Temperature	[29]	Assigned
Latest commit number of files	latest_commit-Number_of_files	External Temperature	[60]	Assigned
Temporal Locality	temporal_locality	External Temperature	[26] [45]	Open
Temporal Locality: Weighted distance	temporal_locality-weighted	External Temperature	[26] [45]	Open
Number of Negative sentiment	nlp4re_sentiment-CM_NNS	Internal Temperature	[94]	Open
Percentage of negative sentiment	nlp4re_sentiment-CM_PNS	Internal Temperature	[94]	Open
Activities comments	activities-comments_count	Internal Temperature	[7]	Open
Activities count	activities-count	Internal Temperature	[7]	Open
Activities histories	activities-histories_Count	Internal Temperature	[7]	Open
Activities work items count	activities-work_items_Count	Internal Temperature	[7]	Open
One negative sentiment	nlp4re_sentiment-CM_ONS	Internal Temperature	[94]	Open
Sentiment Polarity	nlp4re_sentiment-IT_POL	Internal Temperature	[7] [105]	Open
Sentiment Subjectivity	nlp4re_sentiment-IT_SUB	Internal Temperature	[7] [105]	Open
Ticket Participants Count	issue_Participants-count	Internal Temperature	[79]	Open
Action density	nlp4re_description-EX_ACD	Intrinsic	[52]	Open
Components max bugginess	components-Max_Bugginess	Intrinsic	[77]	Open
Component Count	components-Count	Intrinsic	[77]	Open
Description Attribute: Actions	nlp4re_description-DA_ACT	Intrinsic	[52]	Open
Description Attribute: Conditionals	nlp4re_description-DA_CND	Intrinsic	[99, 55]	Open
Description Attribute: Continuances	nlp4re_description-DA_CNT	Intrinsic	[99, 55]	Open
Description Attribute: Imperatives	nlp4re_description-DA_IMP	Intrinsic	[99, 55]	Open
Description Attribute: incompletes	nlp4re_description-DA_INC	Intrinsic	[99, 55]	Open
Description Attribute: Options	nlp4re_description-DA_OPT	Intrinsic	[99, 55]	Open
Description Attribute: Risk Level	nlp4re_description-DA_RKL	Intrinsic	[52] [99, 55]	Open
Description Attribute: Sources	nlp4re_description-DA_SRC	Intrinsic	[52]	Open
Description Attribute: Weak Phrases	nlp4re_description-DA_WKP	Intrinsic	[99, 55]	Open
Number of Ambiguities	nlp4re_description-EX_AMG	Intrinsic	[52]	Open
Number of Directives	nlp4re_description-EX_DIR	Intrinsic	[99]	Open
Number of Entities	nlp4re_description-EX_ENT	Intrinsic	[52]	Open
Number of subjects	nlp4re_description-EX_SBJ	Intrinsic	[52]	Open
Number of words	nlp4re_description-EX_CNS	Intrinsic	[99]	Open
Number of Verbs	nlp4re_description-EX_VRB	Intrinsic	[52]	Open
Priority	priority	Intrinsic		Open
Readability Score	nlp4re_description-EX_RDS	Intrinsic	[99]	Open
Sentences Completeness	nlp4re_description-EX_ICP	Intrinsic	[99]	Open
Type	type	Intrinsic	[46] [100]	Open
Average Bag of Words Cosine Text similarity with previous buggy tickets	buggy_similarity-AvgSimilarity_BagOfWords_Cosine_Text	R2R	[49]	Open
Average TF-IDF Cosine title similarity with previous buggy tickets	buggy_similarity-AvgSimilarity_TFIDFCosine_Title	R2R	[82]	Open
Max Levenshtein title similarity with previous buggy tickets	buggy_similarity-MaxSimilarity_Levenshtein_Title	R2R	[66]	Open

3.1 Code

Obviously, the code base on which a ticket is implemented impacts the bugginess of the ticket; in other words, the same ticket could lead to a bug according to how easy the code base is to accept its implementation. This concept has been largely studied in the area of software maintainability [16] [26].

3.1.1 Code Quality

A project littered with badly-written code can lead to a higher number of bugs [16] [26].

Number of Smells *(code_quality-Smells_count)* We take into account the following aspects regarding software quality: Design, Best Practices, Code Style, Documentation, Error-Prone Handling, Multithreading, and Performance. We did this by using PMD [20], a source code quality analysis tool. The tool applies a set of rules [80] to the code base and reports each violation. We count each violation as a code smell [34]. The rules can be divided in the following categories:

- Design: we took into consideration Abstract Class Without Any Method, Class With Only Private Constructors Should Be Final, Do Not Extend Java Lang Error, Final Field Could Be Static, Logic Inversion, Simplified Ternary, Simplify Boolean Returns, Simplify Conditional, Singular Field, Useless Overriding Method, and Use Utility Class.

- Best Practices: we focused on Avoid Message Digest Field, Avoid String Buffer Field, Avoid Using Hard Coded IP, Check Result Set, Constants In Interface, Default Label Not Last In Switch, Double Brace Initialization, For Loop Can Be Foreach, Guard Log Statement, Literals First In Comparisons, Loose Coupling, Missing Override, Non Exhaustive Switch, One Declaration Per Line, Primitive Wrapper Instantiation, Preserve Stack Trace, Simplifiable Test Assertion, Unused Formal Parameter, Unused Local Variable, Unused Private Field, Unused Private Method, Use Collection Is Empty, and Use Standard Charsets.

- Code Style: we considered Class Naming Conventions, Formal Parameter Naming Conventions, Generics Naming, Lambda Can Be Method Reference, Local Variable Naming Conventions, Method Naming Conventions, Package Case, Avoid Dollar Signs, Avoid Protected Field In Final Class, Avoid Protected Method In Final Class Not Extending, Control Statement Braces, Extends Object, Final Parameter In Abstract Method, For Loop Should Be While Loop, Identical Catch Branches, No Package, Unnecessary Annotation Value Element, Unnecessary Constructor, Unnecessary Fully Qualified Name, Unnecessary Import, Unnecessary Local Before Return, Unnecessary Modifier, Unnecessary Return, Useless Parentheses, and Useless Qualified This.

- Documentation: we used Uncommented Empty Constructor and Uncommented Empty Method Body.

- Error-Prone Handling: we applied Assignment In Operand, Assignment To Non Final Static, Avoid Accessibility Alteration, Avoid Branching Statement As Last In Loop, Avoid Catching Throwable, Avoid Decimal Literals In BigDecimal Constructor, Avoid Instanceof Checks In Catch Clause, Avoid Multiple Unary Operators, Avoid Using Octal Values, Broken Null Check, Check Skip Result, Class Cast Exception With ToArray, Clone Method Must Be Public, Clone Method Must Implement Cloneable, Clone Method Return Type Must Match Class Name, Close Resource, Compare Objects With Equals, Comparison With NaN, Do Not Call Garbage Collection Explicitly, Do Not Extend Java Lang Throwable, Don't Use Float Type For Loop Indices, Equals Null, Idempotent Operations, Implicit Switch Fall Through, Instantiation To Get Class, Jumbled Incrementer, Misplaced Null Check, Missing Static Method In Non Instantiatable Class, Non Case Label In Switch, Non Static Initializer, Override Both Equals And Hashcode, Proper Clone Implementation, Proper Logger, Return Empty Collection Rather Than Null, Return From Finally Block, Single Method Singleton, Singleton Class Returning New Instance, Suspicious Equals Method Name, Suspicious Hashcode Method Name, Suspicious Octal Escape, Unconditional If Statement, Unnecessary Conversion Temporary, Unused Null Check In Equals, Use Equals To Compare Strings, Useless Operation On Immutable, and Use Locale With Case Conversions. Regarding Multithreading, we included Avoid Thread Group, Avoid Using Volatile, Don't Call Thread Run, Double Checked Locking, Non Thread Safe Singleton, Unsynchronized Static Formatter, and Use Notify All Instead Of Notify. Regarding Performance, we took into account Big Integer Instantiation and Optimizable To Array Call.

- Multithreading: we included Avoid Thread Group, Avoid Using Volatile, Don't Call Thread Run, Double Checked Locking, Non Thread Safe Singleton, Unsynchronized Static Formatter, and Use Notify All Instead Of Notify.
- Performance: we took into account Big Integer Instantiation and Optimizable To Array Call.

3.1.2 Code Size and Complexity

A codebase with a huge size and complexity can be hard to maintain and understand correctly.

Total LOCs (*code_size-total_LOCs*): One of the first complexity measure is size [104]. We measure code size by counting the number of lines of code (LOCs) in the project.

Number of files (*code_size-number_of_files*): We count the number of files in the project to further measure the size of the code base.

Number of different languages (*code_size-number_of_languages*): We count the number of different languages used across the project files, since the higher the number of languages, the higher the defect proneness [61].

3.2 Developer

We take into account the developer assigned to the ticket, namely the developer tasked to implement the changes described in the ticket.

ANFIC (*assignee-ANFIC*): This features was introduced by Matsumoto et al. [71] to measure the average number of bug injected by a developer per commit. We transfer the same concept to the ticket level, by measuring the number of bug-inducing issues historically assigned to the developer divided by all the issues assigned to them.

Familiarity (*assignee-Familiarity*): A developer familiarity with the code they are working on impacts the quality of their work [96]. We measure the assigned developer familiarity by counting how many issues have been historically assigned to them to divided by the total number of issues in the project.

3.3 External Temperature

Implementing a ticket in an unstable environment can be hard. The features belonging to the External Temperature family take into account how often the project is subject to changes.

Temporal Locality (*temporal_locality*): Since one bug can lead to another [45], we want to measure if the ticket is involved in a "hot" time span, when many bugs seem to happen close in time [26]. We measure the proportion of bug-inducing issues among all issues prior to the measured issue in a limited time horizon.

Weighted Temporal Locality (*temporal_locality-weighted*): We further expand the Temporal Locality concept by weighting the bug-inducing issues the more they are close in time to the measured issue.

Number of Commits while in progress (*commits_while_in_progress-Count*): Projects subject to parallel work have a higher number of quality problems [78]. We measure the number of commits submitted to the project while the ticket was in progress.

Churn of Commits while in progress (*commits_while_in_progress-Churn*): Following the same reasoning endorsing the previous feature, we measure the total number of LOCs changed by commits submitted to the project while the ticket was in progress. A LOC is considered changed if it was added, modified or deleted.

Latest Commit Churn (*latest_commit-Churn*): Software subject to many changes is more likely to incur in bugs [29]. We measure the total number of LOCs changed by the most recent commit in the project previous to the measured ticket.

Latest Commit Number of Files (*latest_commit-Number_of_files*): We measure the number of files changed by the most recent commit in the project previous to the measured ticket. This feature is inspired by the Entropy feature described in Keshavarz and Nagappan [60], since it aims to capture the dispersion of the changes.

3.4 Internal Temperature

Some file level bug-predicting approaches assume that files recently or frequently changed are more bug-prone [22]. We transfer this concept to the ticket level by measuring the "hotness" of the ticket, namely when, how and how often the ticket was changed.

Participants Count (*issue_Participants-count*): The more the developers working on a software module, the higher the chance a defect is injected as a result [79]. We measure the number of participants involved in the ticket implementation, i.e. authors of changelog entries, reporter, assignee, creator.

Activities Count (*activities-count*): Developers tend to discuss problematic software entities more [7]. We transfer this concept to the ticket level by counting ticket Activities. Activities in a ticket can be comments, work items and histories.

Comments count (*activities-comments*): Ticket participants use comments to express their opinions, ask for clarifications, provide additional information, etc.

Work Items count (*activities-work_items_Count*): Work items are used to track the time spent by participants on the ticket.

Histories count (*activities-histories*): Histories are used to track the changes made to the ticket.

3.4.1 NLP for RE - Sentiment

The subfamily takes into consideration the sentiment analysis applied to requirement description and comments. The sentiment analysis has been extensively studied by Zhang and Liu [105]: for our purpose, we solely focused on the aspects of *polarity* and *subjectivity* associated to a sentence, stemming from the idea that using subjective language for describing requirements could induce defects in design and implementation [52]. Particularly, we build upon the findings of Bacchelli, D'Ambros, and Lanza [7], which recommend the use of the "popularity" metric as an indicator of potential defect introduction, and those of Valdez et al. [94], which demonstrate a correlation between comment sentiment and factors as bug resolution speed and SW Professional's quality in tasks, by introducing "occurrence of negative comments" and "percentage of negative comments relative to total comments" as more representative features for identifying potential defect introduction.

Sentiment polarity (*IT_POL*): measures the positiveness (or negativeness) of a sentence. It is a number lying between -1 (extremely negative) and 1(extremely positive) [105].

Sentiment subjectivity (*IT_SUB*): measures the amount of personal opinion and feelings with respect to factual information contained in a sentence. Typically, the higher the index, the less objective is the language used in a sentence [105]. It is a number lying between $[0; 1]$.

Number of negative comments (*CM_NNS*): measures the occurrence of negative comments associated to a requirement

Percentage of negative comments (*CM_PNS*): measures the occurrence of negative comments divided by the total number fo comments

Presence of one negative comment(*CM_ONS*): measures the presence of at least one negative comment

3.5 Intrinsic

Intuitively, some tickets can be considered inherently more difficult to implement than others.

Priority (*priority*): We measure the priority of the ticket, namely a level of importance telling what ticket should be implemented first. Prioritizing one ticket over another means allocating more time to the former at the cost of the latter, hence the latter could be subject to a rushed development which could produce a buggy implementation. Besides, the higher the priority, the more urgent the ticket is, the more the stress can burden the assignee, leading to a higher chance of mistakes.

Components Count (*components-Count*): This feature is inspired by the Entropy feature described in Patel, Adams, and Hassan [77]. We count the number of project components the ticket is related to, in order to measure the dispersion of the required changes.

Components Max Bugginess(*components-Max_Bugginess*): We measure the highest bugginess among the components the ticket is related to. We compute the bugginess of the component by counting the number of bug-inducing ticket historically related to the component divided by the total number of tickets related to the component until the measurement date. The idea is that a historically buggy component could be inherently fragile, hence further changes to it could induce more bugs.

Type(*type*): It has been shown that often bug fixing changes induce more bugs in the code [46]. We extend this concept by considering the change type of the ticket, i.e: bug, improvement, new feature, subtask, etc. It is worth noting than the activity of finding and fixing bugs, although is the most rewarding when successful, can be very frustrating and stressful [100], leading to a higher chance of mistakes.

3.5.1 NLP for RE - Description

The subfamily takes into consideration some aspects of the syntax and semantic analysis of the requirement title and description. We share the approach adopted by [99] and [55], but, due to the intrinsic nature of our requirement datasets (i.e. Jira tickets) we applied (some of the) entire-document-related indicators directly to requirement text. We extended the approach by including the analysis of "actions", "named entities" and reference to "external resources", as metrics associated to the complexity requirement complexity [52].

Description attribute action *(DA_ACT)*: measures the occurrence of actions by applying patterns to detect obligations and compound verb phrases [52]. The higher, the more probable is that the current requirement should be split into multiple requirements.

Description attribute conditionals *(DA_CND)*: measures the occurrence of conditional patterns in a sentence, such as the presence of words like "if," "when," "unless," and "depends on," among others [99, 55]

Description attribute continuances *(DA_CNT)*: measures the occurrence of continuance indicators in a sentence, including phrases like "see below," "as follows," "listed," and "in particular," among others [99, 55]

Description attribute imperatives *(DA_IMP)*: measures the occurrence of imperative expressions in a sentence, such as "shall," "must," and "is required to," among others [99, 55]

Description attribute incompletes *(DA_INC)*: measures the occurrence of incomplete markers, identifying acronyms like "TBD," "TBR," "TBC," and "TODO," which indicate missing or pending content [99, 55]

Description attribute options *(DA_OPT)*: measures the occurrence of the use of optionality markers, including words like "can," "could," "may," and "optionally," which indicate non-mandatory elements [99, 55]

Description attribute sources *(DA_SRC)*: measures the occurrence of references to external resources, like files and websites [52]

Description attribute weak phrases *(DA_WKP)*: measures the occurrence of vague or non-assertive phrases that may weaken the clarity and precision of a sentence, e.g. ”adequate”, ”as a minimum”, ”be capable of” and so on [99, 55]

Description attribute risk level *(DA_RKL)*: measures the overall level of risk associated to each requirement by summing up each previous index ”DA_¡¿”

Number of subjects *(EX_SBJ)*: measures the number of general nouns in sentence, identifying subjects and objects [52]

Number of words *(EX_CNS)*: measures the number of words in sentence [99]

Number of verbs *(EX_VRB)*: measures the occurrence of verbs [52]

Number of ambiguities *(EX_AMG)*: measures the number of ambiguos words used in the sentences, e.g. ”some”, ”many”, ”few”, ”often” and so on. [52]

Number of directives *(EX_DIR)*: measures the use of directives markers that represent instructions or references, such as ”e.g.,” ”i.e.,” ”figure,” ”table,” ”for example,” and ”note.” [99]

Readability score *(EX_RDS)*: measures how readable a piece of text by applying the ”Flesch reading ease” score. The lowest is the score, the more technical is the language used in the sentence. [99]

Sentence completeness *(EX_ICP)*: measures whether a sentence is syntactically complete based on the presence of a nominal subject, a verb, and an object (direct, indirect, or prepositional) [99]

Action density *(EX_ACD)*: measures the number of actions with respect the total number of words [52]

Number of entities *(EX_ENT)*: measures the number of recognized named entity (NER) in a sentence [52]

3.6 Requirement to Requirements

It is intuitive that tickets that are more semantically similar to tickets that induced a bug are more prone to induce a bug. Therefore, the idea behind this family of features is to use the level of similarity to past bug-inducing tickets as a feature for TLP. To measure the semantic similarity we took into consideration three natural language processing (NLP) techniques, two aggregation techniques and two inputs. Regarding the NLP techniques, we took into consideration:

- Levestein: also known as edit distance, measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into another. It is effective for detecting textual similarities and variations [66].
- BagOfWords: represents text as an unordered collection of words, disregarding grammar and word order while preserving word frequency. It converts text into numerical vectors by counting word occurrences, making it useful for text classification, document similarity, and feature extraction tasks. Although simple, BoW is effective in many NLP applications but lacks semantic understanding due to its inability to capture word relationships [49].
- TF-IDF: is a statistical measure of the importance of a word in a document relative to a collection (corpus). It is computed as the product of term frequency (TF), i.e., the word occurrences in a document, and inverse document frequency (IDF), i.e., the weight of frequently occurring words in the corpus [82].

Since the NLP techniques above provides a similarity score for each pair of texts, we need a technique that aggregates for a ticket the similarity score of each previously bug-inducing tickets. Regarding aggregation techniques, we took into consideration:

- Max: it selects the highest similarity scores with previous bug-inducing tickets. The rationale is that tickets very similar to a single bug-inducing ticket are likely bug-inducing, regardless of how much these tickets are not similar to other bug-inducing tickets.

- Average: it measures the average similarity scores with previous bug-inducing tickets. The rationale is that tickets very similar to the set of bug-inducing tickets are likely bug-inducing, regardless of how much these tickets are similar to a specific bug-inducing ticket.

Since the NLP techniques requires two texts as input to compute their similarity score, we considered as text the title of the ticket and the description of the ticket.

The total number of possible R2R features is 12, i.e., three NLP techniques * two aggregation techniques * two inputs. However, since these 12 are intrinsically correlated, we chose three out of the twelve with the aim of considering at least one for each type of NLP technique, aggregation and input: Levenshtein_Max_Title, TFIDF_Avg_Title, BagOfWords_Avg_Description.

3.7 JIT

We consider the features of the commits implementing the ticket when they are available according to the measurement date. In this study, we considered the commit features described by Kamei et al. [58] and Keshavarz and Nagappan [60]. We note that we neglected the feature "Year" since it is already available as a feature Author date. Since a Ticket can be linked to several commits, we aggregated the above-mentioned features of all linked commits of a ticket by using an Aggregation strategy specific for each feature in order to capture the same information brought by the per-commit feature.

Table 2: JIT features. There are a total of 15 features.

Name	Definition	Sources	Aggregation
ns	number of modified subsystems	[58] [60]	MAX
nd	number of modified directories	[58] [60]	MAX
nf	number of modified files	[58] [60]	MAX
ent	Distribution of modified code across each file	[58] [60]	MAX
la	Lines of code added	[58] [60]	SUM
ld	Lines of code deleted	[58] [60]	SUM
lt	Line of code in a file before the change	[58] [60]	SUM
fix	Whether or not the change is a defect fix	[58] [60]	COUNT(True)
ndev	Number of developers that changed the modifed files	[58] [60]	MAX
age	The average time interval between the last and the current change	[58] [60]	MIN
nuc	number of unique changes	[58] [60]	MAX
aexp	Developer experience	[58] [60]	MIN
arexp	Recent developer experience	[58] [60]	MIN
asexp	Developer experience on a subsystem	[58] [60]	MIN
author_date	change date	[58] [60]	MAX(date) - MIN(date)
num_commits	Number of commits linked to the ticket	[23]	COUNT

The JIT features are summarized in Table 2.

Authors Count (*jit-ndev-MAX*): We count the highest number of authors involved in a commit linked to the ticket.

Developer Recent Experience (*jit-arexp-MIN*): We measure the lowest recent experience of the authors involved in the commits linked to the ticket.

Developer Experience (*jit-aexp-MIN*): We measure the lowest experience of the authors involved in the commits linked to the ticket.

Developer Subsystem Experience (*jit-asexp-MIN*): We measure the lowest subsystem experience of the authors involved in the commits linked to the ticket.

Modified Subsystems Count (*jit-ns-MAX*): We count the highest number of subsystems modified by a commit linked to the ticket.

Age (*jit-age-MIN*): We measure the lowest temporal distance between a commit linked to the ticket and the most recent commit preceding it.

Author Date (*jit-author_date-DURATION*): We measure the time span between the earliest and the latest commit linked to the ticket.

LOCs Added (*jit-la-SUM*): We sum the LOCs added by all commits linked to the ticket.

LOCs Deleted (*jit-ld-SUM*): We sum the LOCs deleted by all commits linked to the ticket.

Type (*jit-fix-COUNT_TRUE*): We count how many fixing commits are linked to the ticket. Fixing changes are more prone to introduce bugs [46].

Modified Directories Count (*jit-nd-MAX*): We count the highest number of directories modified by a commit linked to the ticket.

Unique Changes Count (*jit-nuc-MAX*): We count the highest number of unique changes made by a commit linked to the ticket.

Entropy (*jit-ent-MAX*): We measure the highest entropy of a commit linked to the ticket.

Modified Files Count (*jit-nf-MAX*): We count the highest number of files modified by a commit linked to the ticket.

Number of Commits (*num_commits*): we count the number of commits linked to the ticket to distinguish the cases when commits with different values for every JIT feature produce same values when aggregated.

4 Study Design

4.1 RQ1: Does temporal proximity impact the accuracy of TLP?

4.1.1 Introduction

Knowing the future before it happens can be of great importance, but knowing it in time to act upon it arguably has the most value. In fact, The earlier we can make an accurate prediction, the more efficiently we can cope with its outcomes. This RQ explores the impact of shifting left the prediction moment about the presence of bugs, with the aim to evaluate the classifiers' performance tasked to predict if a ticket will induce a bug at different stages of its lifecycle, which is described in Figure 2. It is worth noting that earlier stages of the ticket lifecycle could yield less data to train the predictors, both in quality and quantity, actually damaging the predictions' accuracy, while gaining in anticipation and thus in the possibility to act upon the prediction (i.e: plan more test to assess the correctness of the implementation of the ticket).

4.1.2 Independent Variables

The independent variable of RQ1 is the temporal proximity of TLC. This variable has three treatments, i.e., proximity points, which reflect the lifecycle of project development (as shown in Figure 3):

- Before Ticket Assignment, aka, **Open**: The ticket is created but not yet assigned.
- Before First Commit, aka, **InProgress**: The ticket is assigned, and no commit has been submitted yet. We measure this period starting from the date of the first assignment (as provided in JIRA) until one second before the first commit (as provided in Git).
- After Last Commit, aka, **Closed**: The ticket is implemented. We measure this as the time of the last commit related to the ticket.

4.1.3 Dependent Variables

The main dependent variable is the accuracy of TLP. As performance indicators of TLP we used the following six metrics which are standards in ML:

- AUC: Area Under the Receiving Operating Characteristic Curve [25] is the area under the curve of true positive rate versus false positive rate, which is

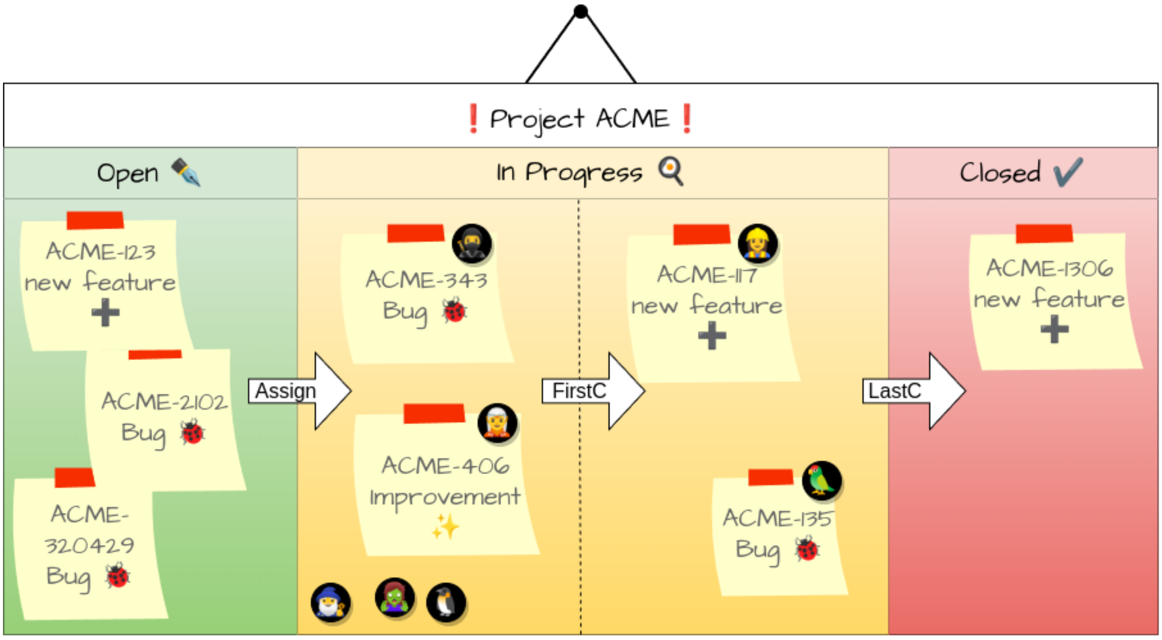


Figure 2: Issue lifecycle

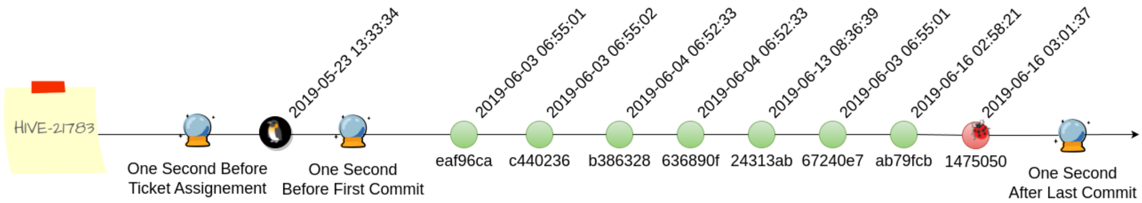


Figure 3: Measurement dates

defined by setting multiple thresholds. A positive instance is a bug-inducing ticket, whereas a negative instance is a non bug-inducing ticket. AUC has the advantage of being threshold independent and, therefore, it is recommended for evaluating defect prediction techniques [65];

- Precision: It is an accuracy metric describing how often the positive predictions of the model are correct [26];
- Recall: It describes how often the model predicts true positive entities as actually positive [26].
- Kappa: It describes how well the predictor performs compared against a random guesser [23];
- Specificity: It is a concept close to Recall, but it is related to the negative class. It describes how often the model predicts negative entities as actually negative [77];
- GMean: When used to evaluate models performing binary classification tasks, it is defined as the square root of the product between recall and specificity [77].

4.1.4 Hypotheses and testing

Our null hypothesis H01 is that the accuracy of TLP does not vary across temporal points, i.e., temporal proximity does not impact TLP accuracy.

To assess the statistical significance of differences between the treatments, we employed the Wilcoxon Signed-Rank Test, a non-parametric test suitable for paired data [98]. This test is advantageous as it does not assume a normal distribution of the differences, making it robust for datasets with non-normal distributions or small sample sizes. In this study, each accuracy metric was tested for significant differences across treatments within the same dataset and classifier. We set alpha to 0.05 [41].

4.1.5 Performance Measurement

We chose the following supervised machine learning models since they have been used in JIT prediction studies [77]:

- Random Forest (RF): This model is an ensemble learning method that consults a random subset of decision trees whenever it makes a prediction in order to reduce correlation among the bagged trees [54];

- Logistic Regression (LR): It is a variant of the linear regression model specialized in binary classification tasks [54];

- Neural Network (NN): It is a model inspired by the human brain that is able to learn complex patterns in the data by fitting the weights of the connections between its neurons, which are organized in layers and exchange information through activation functions [54];

- AutoWEKA (AW): It is a metamodel that automatically selects the best classifier and its hyperparameters for a given dataset with the aim to maximize a given performance metric [62].

A dimension of evaluation is the feature selection technique, since the presence of useless features can hinder the model learning process. We used the following techniques:

- None: No feature selection is applied. We use this case as a baseline in order to see if the application of feature selection actually improves the classifiers performances;

- Forward: It is a model-wise approach that greedily builds a subset of features by starting from an empty set and adding features which are highly correlated with the target and not with the already selected features, while seeing if the newly produced set of features improves the classifier performances [101];
- Backward: It is a model-wise approach that greedily builds a subset of features by starting from the full set and removing features which are not highly correlated with the target, while seeing if the newly produced set of features improves the classifier performances [101];
- Filter: It is a model-independent approach that builds a subset of features by selecting the ones which are highly correlated with the target and have little correlation among them [101].

We compared the use of SMOTE as a balancing technique [18] against the absence of it. Balancing the population numbers of both bug inducing and non bug-inducing ticket can help the models to learn better from the dataset. We did this comparison following the work of Patel, Adams, and Hassan [77].

As validation technique we used two techniques: moving window (Figure 5) and 80-20 ordered holdout (Figure 4). This approach deviates from the common practice in software engineering research, where typically only a single validation technique is utilized (e.g., [77]) and it aims to enhance the robustness of our findings and leverage the pros of both techniques. Specifically, the pros of sliding window are that it produces more data points in order to extract the distributions of the results, however it is computationally expensive and uses less data at each window. The pros of 80-20 ordered holdout are that it leverages the full dataset to train the models, however it fails to capture pitfalls such as the concept drift.

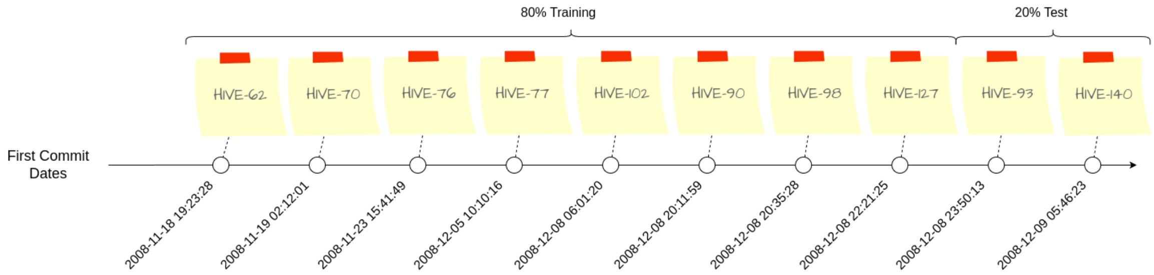


Figure 4: 80-20 Ordered Holdout example using the first commit date as measurement date.

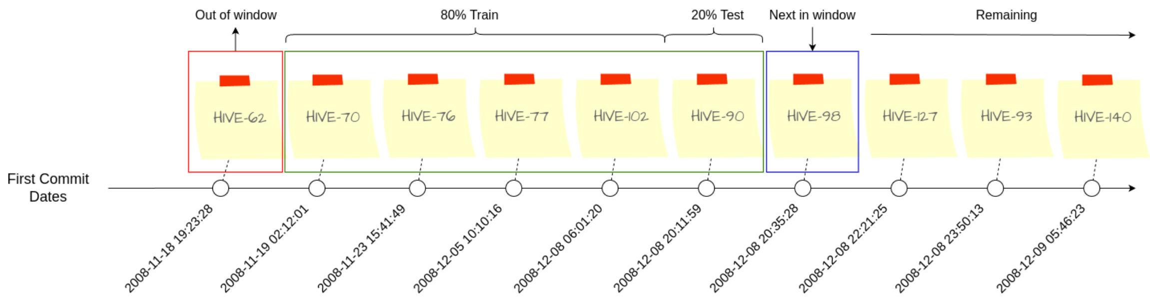


Figure 5: Sliding Window example using the first commit date as measurement date.

It is worth noting that we don't use AW in the Sliding Window design since it is not feasible to run it in a reasonable time frame.

4.2 RQ2: Does temporal proximity impact the power of TLP features?

4.2.1 Introduction

Some features can be harder to measure than others, especially when such features depend on humans annotating metadata on the ticket (i.e.: assignments). Besides, the more features we use, the more data we need to train our models. This problem is known in the Machine Learning landscape as the Curse of Dimensionality [21]. For the mentioned reasons, a project manager could be interested in which features are both easier to measure and most informative. This RQ explores how the importance of each feature is impacted by shifting left the moment of the prediction in the ticket lifecycle.

4.2.2 Independent Variables

The independent variables of RQ2 are the temporal proximity of TLC and the features used for TLC. The temporal proximity has the same three treatments described in RQ1. The features used for TLC are the 62 detailed in section 3.

4.2.3 Dependent Variables

A dependent variable is the Information Gain Ratio (IGR), since it has been used in a similar work by Falessi et al. [26]. We use this metric since it is agnostic in respect to the classifiers. However, the downside is that specific values of IGR are hard to interpret without a given reference [26].

Another dependent variable is the Feature Selection (FS) result, namely a binary variable telling for each configuration if the feature was selected or not.

4.2.4 Hypotheses and testing

Our null hypothesis H20 is that the power of TLP features does not vary across feature family, temporal points, and their interaction.

We applied the same statistical tests of RQ1.

4.2.5 Performance Measurement

In order to produce the evaluation results we applied the same techniques described for RQ1.

4.3 Measurement Procedure

In order to produce the results, we followed the approach illustrated in Figure 6. First things first, we needed to find some reusable datasets in order to train and test the

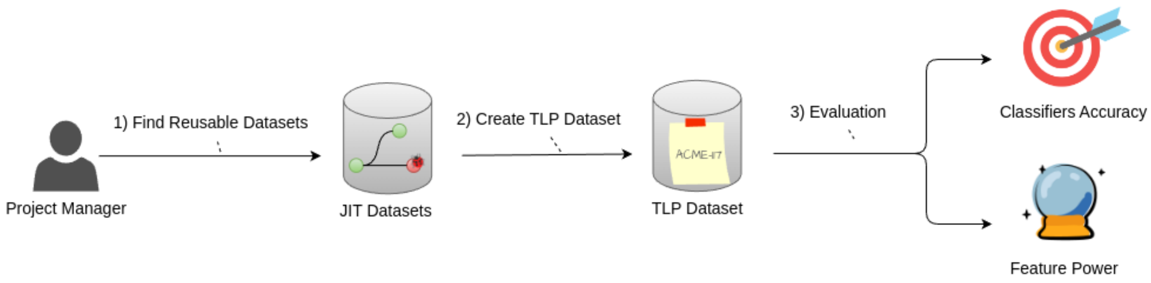


Figure 6: Phases overview

models. Then, we created the TLP datasets to actually feed the models. Finally, we produced and analyzed both the results of classifiers accuracy and feature power evaluation, answering respectively RQ1 and RQ2.

4.3.1 Find reusable datasets

To the best of our knowledge there are no publicly available datasets that extract the features described in section 3 and label the tickets as bug-inducing and non bug-inducing. However, we did find some JIT datasets, reporting hashes and features of commits labeled as buggy or not [15] [25] [60].

4.3.2 Create TLP datasets

In order to create the TLP dataset, we followed the steps illustrated in Figure 7. Starting from the JIT datasets, we downloaded the tickets linked to the commits following the steps described in algorithm 1, thus obtaining a set of tickets along with their linked tickets. We labeled the tickets as bug-inducing according to a JIT dataset if they had at least one linked commit labeled as buggy. It is worth noting that different JIT dataset could consider the same commit as buggy or not, depending on the criteria used to label the commits. From now on, we use the term "Project" referring to the pairing of a JIT dataset and a Jira project (i.e: apachejit_HIVE). An

Algorithm 1 Set of JIT Datasets $\rightarrow$ Set of linked Tickets

- for each JIT Dataset:
  - for each commit reported in the JIT Dataset:
    1. load the commit with its coordinates and JIT features;
       * Commit coordinates include the commit hash and repository.
    2. Get the commit message from the corresponding repository log;
       * If the repository is not available, it is cloned from the corresponding remote host;
       * The projects considered in this study are all versioned using Git and hosted on GitHub.
    3. Scan the commit message for ticket key mentions;
       * An ticket key is a unique identifier for an ticket in a project management system.
       * The format of the key is specific to the project management system.
       * All projects considered in this study use Jira as their project management system.
    4. For each mentioned ticket key:
       (a) Search for matching issues in the project management system;
       (b) For each matching ticket found:
           i. Download ticket details, which include fields and changelog;
              * ticket fields include, but are not limited to: title, description, resolution, due date, opening date, watchers, comments, worklog, votes, attachments, linked issues, subtasks, status, priority, type, assignee, reporter, creator, project, components;
              * ticket changelog consists in a time ordered list of changes to the ticket fields, along with the date and the author of the changes.
           ii. Link ticket to commit;
    5. Load commit timestamp from the corresponding repository.
    6. Store commit along with the mentioned issues.

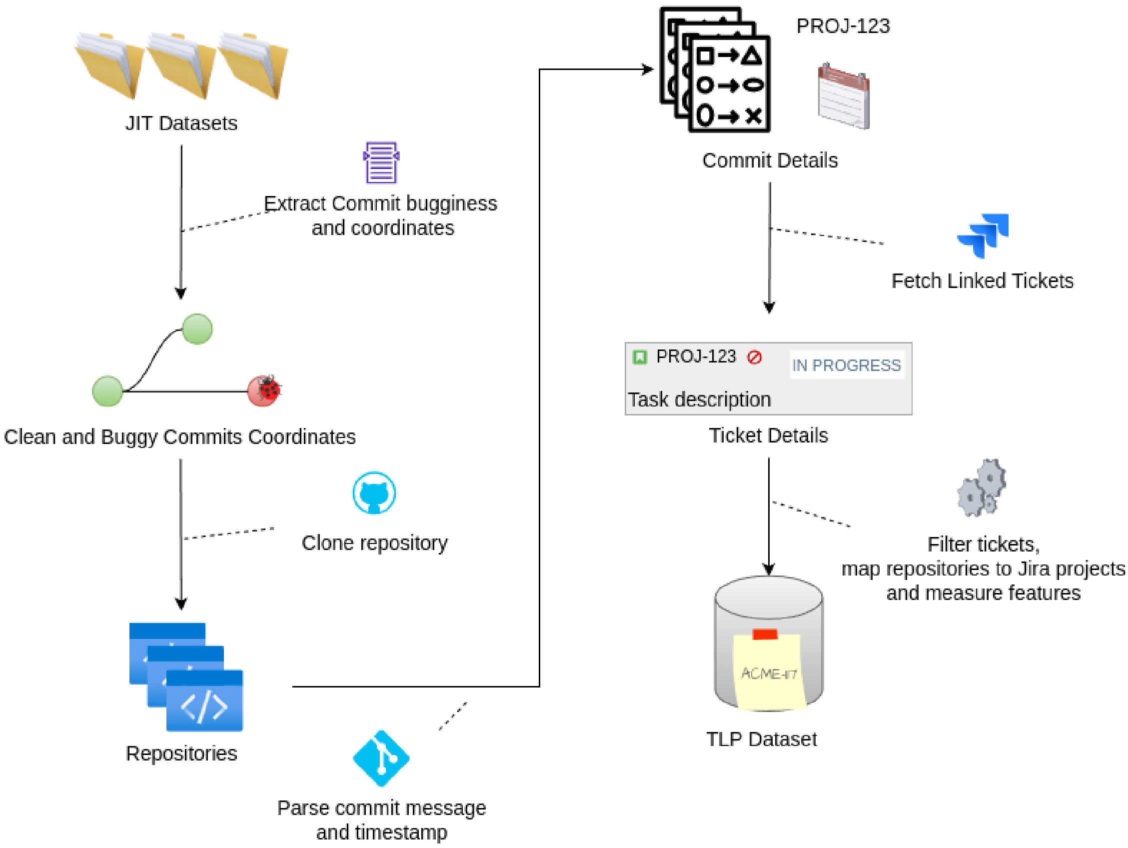


Figure 7: TLP dataset creation overview.

example of a ticket is shown in Figure 8.

The next step was to map projects to repositories in order to access the data related to their codebases. Since a ticket can have multiple commits, it can be not so obvious to guess the main repository of a project. In order to guess if a repository is the main one for a project, we test the following conditions:

- the repository has the highest number of commits linked to tickets of that project;
- the project has the highest number of tickets linked by commits from that repository.

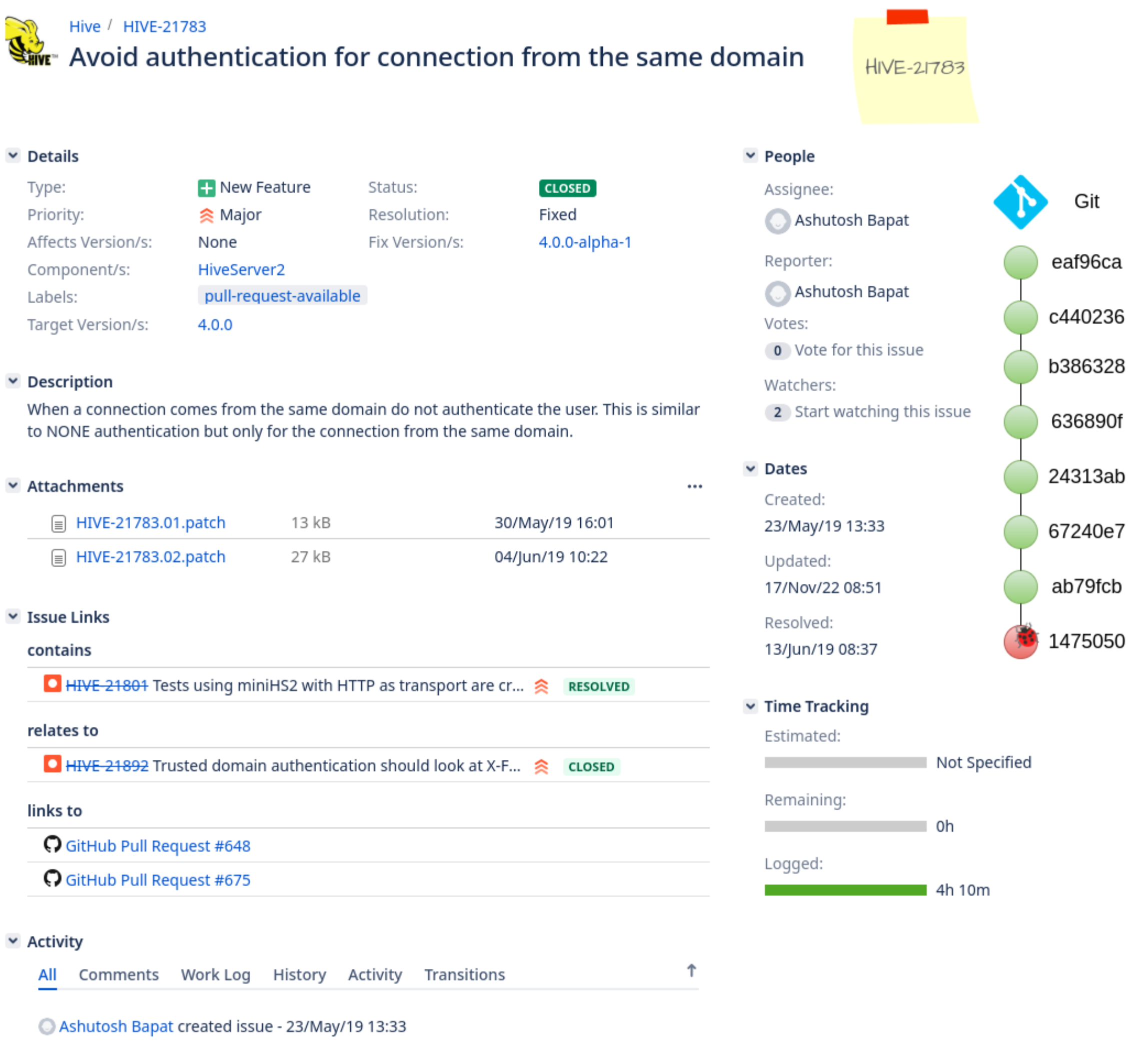


Figure 8: Ticket Example

To clarify, consider the following made-up example. Let apache/camel be a repository which 95% of commits are linked to the Camel Jira Project, while the remaining 5% is linked to the Zeppelin Jira Project. Following the same reasoning, let apache/zeppelin be a repository which 99% of commits are linked to the Zeppelin Jira Project, while the remaining 1% is linked to the Apache Common Jira Project. According to the first condition we get the following pairings:

- apache/camel ↔ Camel;
- apache/zeppelin ↔ Zeppelin;
- apache/zeppelin ↔ Apache Common.

Clearly Apache Common cannot have zeppelin as its main repository since only 1% of its linked commits point to Apache Common tickets. Nevertheless, apache/zeppelin is the repository with the most commits linked to Apache Common tickets in the dataset. The second condition comes into play by ruling out the pairing between apache/zeppelin and Apache Common, leaving only the one between apache/zeppelin and Zeppelin since it has the highest percentage of links from that repository. The final mapping would be the following:

- apache/camel ↔ Camel;
- apache/zeppelin ↔ Zeppelin.

It is worth noting that, according to the above-mentioned procedure, there can be projects which have no guessed repository.

Afterward, we filtered anomalous tickets since they can confuse the models. In order to spot anomalous tickets we applied the following filter:

- `ExclusiveBuggyCommitsOnlyFilter`: This filter discards all tickets which have at least one buggy commit and all its buggy commits are shared with other tickets, since we cannot say for sure which ticket induced the bug;
- `FirstCommitAfterOpeningDateFilter`: This filter discards all tickets having the date of the first commit after the Opening Date of the ticket;
- `HasGuessedRepositoryFilter`: This filter discards all tickets belonging to a project for which it has been impossible to guess the main repository. This can happen if all the repositories extracted from a JIT dataset have already been paired with a better suitable project;
- `NotMostRecentFilter`: When considering projects coming from JIT datasets ApacheJIT and JITSdp, we discard the most recent 20% of the tickets to mitigate the impact of snoring [23]. We leave LeveragingJIT as is since a similar filtering step has already been applied while building the JIT dataset;
- `MeasurementAfterOpeningDateFilter`: This filter discards all tickets having their measurement date before the Opening Date.

Besides, we exclude from the measurement all the tickets which we were unable to extract a value for a measurement date (i.e.: If a ticket has no assigned developer in its history, we cannot extract the date of the first assignment).

In order to select the projects to consider in this study, we ranked them according to the linkage proportion of buggy tickets, as shown in Table 3. We selected the projects HIVE and HBASE from apachejit since they have the highest buggy linkage while having lots of usable tickets.

Table 3: List of projects considered in the study, ranked by the percentage of buggy commits linked to Jira tickets. Each project is identified by its name and the datasets its commits come from. The usable ticket count is the number of tickets that have survived the filtering process.

Dataset	Project	% Buggy Linkage	% Linkage	# Tickets	# Usable Tickets
leveragingjit	ZooKeeper	100	60	91	91
apachejit	Hive	99	99	6443	5025
apachejit	HBase	97	94	7128	5402
leveragingjit	Tika	96	93	126	124
apachejit	Spark	94	86	1298	631
apachejit	ZooKeeper	93	92	748	559
apachejit	Kafka	83	61	1340	600
apachejit	Camel	83	62	7716	6039
apachejit	Cassandra	81	63	4078	3163
apachejit	Flink	78	48	4265	3295
apachejit	ActiveMQ Classic	74	53	2195	1654
apachejit	Ignite	73	49	2870	2276
apachejit	Zeppelin	73	61	866	291
jitsdp	Camel	67	55	9095	7103
leveragingjit	ActiveMQ Artemis	61	30	133	103
leveragingjit	Qpid	59	47	478	394
apachejit	Groovy	57	39	2612	2004
leveragingjit	Directory ApacheDS	50	15	44	44
leveragingjit	Maven	32	18	255	240
leveragingjit	Nutch	29	24	44	44
leveragingjit	OpenJPA	21	17	69	66
leveragingjit	Groovy	20	17	116	114
apachejit	Hadoop Map/Reduce	17	39	833	661
apachejit	Hadoop HDFS	7	20	2842	2236
apachejit	Hadoop Common	0	99	3249	2586

Table 4: Produced datasets summary.

Project	Proximity Point	# Tickets	% Bug-Inducing
HBASE	Open	2415	55,03
HBASE	inProgress	5403	54,40
HBASE	Closed	5407	54,39
HIVE	Open	1510	69,54
HIVE	inProgress	5024	69,77
HIVE	Closed	5027	69,74

We finally produced the TLP dataset by measuring the features described in section 3, producing a dataset for each combination of project and temporal proximity point, resulting in 6 datasets summarized in Table 4. The entities in the dataset are ordered by the value of the measurement date, since it is important to not use future information to predict the past.

4.3.3 Performance evaluation

To evaluate the performance of the classifiers we followed two distinct designs: 80-20 Ordered Holdout and Moving Window.

80-20% Ordered Holdout For each dataset, we split it into 80% training and 20% testing. We feed the RF, LR and NN with the training set, applying each planned Feature Selection technique (None, Forward, Backward, Filter) and SMOTE vs No SMOTE, resulting in $3 \cdot 4 \cdot 2 = 24$ configurations. For each configuration, we evaluate the models on the testing set and produce the results of the accuracy metrics. Additionally, we feed the training set to the AW metamodel instructing it to get the best AUC in 12 hours. Once AW finishes, we annotate the parameters selected by it. It is worth noting that further resampling of the training and testing set in not allowed since it would break the temporal order of the tickets.

Moving Window For each dataset, we initialize the window taking a batch of the first 1000 instances, and we split it into 80% training and 20% testing. For each model among RF, LR and NN, we apply Feature Selection vs No Feature Selection and SMOTE vs No SMOTE on the training set, resulting in $3 \cdot 2 \cdot 2 = 12$ configurations. For each configuration we carry out the evaluation on the corresponding testing test and produce results of the accuracy metrics. Once the evaluation completes, we update the window by removing the first 200 instances and adding the next 200 in the dataset. The procedure described up to now is repeated until all instances in the dataset have been consumed.

5 Case Study Results

5.1 Does temporal proximity impact the accuracy of TLP?

5.1.1 Validation Technique 1: Sliding Window

Figure 9 shows the distributions of TLP accuracy using moving-window in three proximity points. Table 5 and Table 6 show the average gain across classifiers in TLP accuracy using sliding-window when increasing the proximity.

It is intuitive that shifting left the proximity point decreases the accuracy of the predictions, although they are valuable in practice since they are available earlier. The interesting point of this research to evaluate how much and in which direction the accuracy decreases. According to Table 5 and Table 6, we note that:

- The variation differs across metrics. For instance, in HBASE, from Open to InProgress, Recall and F1 even decrease. This could be due to the different imbalance of the Open dataset, which is smaller than InProgress since it does not contain tickets without the Developer Assigned date;

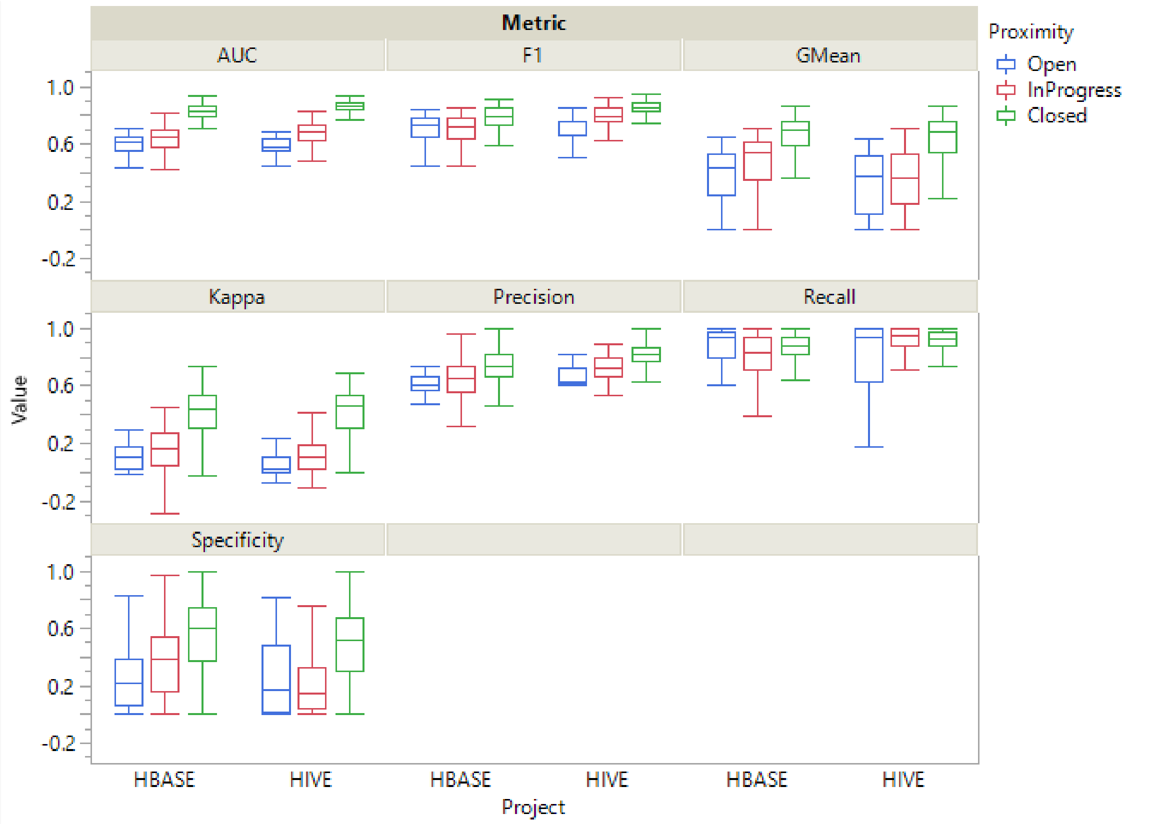


Figure 9: Distributions of TLP accuracy using sliding-window in three proximity points.

Table 5: Average gain across classifiers in TLP accuracy using moving-window in HBASE when increasing the proximity.

	HBASE						
	Precision	**Recall**	**F1**	**AUC**	**GMean**	**Specificity**	**Kappa**
OpenToInProgress	4%	-7%	-2%	6%	44%	45%	22%
InProgressToClosed	15%	8%	13%	29%	154%	49%	42%

Table 6: Average gain across classifiers in TLP accuracy using moving-window in HIVE when increasing the proximity.

	HIVE						
	Precision	**Recall**	**F1**	**AUC**	**GMean**	**Specificity**	**Kappa**
OpenToInProgress	9%	13%	14%	17%	120%	-18%	11%
InProgressToClosed	12%	0%	6%	27%	234%	137%	84%

- The gain from InProgress to Closed is higher than the gain from Open to InProgress, for all seven metrics and both projects;
- Max Kappa value in InProgress is about 0.5 in both projects, showing that the classifiers perform sensibly better than random guessing.

Table 7 reports the statistical test results comparing the accuracy of the same classifier on the same project using moving window over different proximity points. According to Table 7 we can reject H01 in all 24 configurations for the AUC metric.

5.1.2 Validation technique: 80-20% Ordered Holdout

Figure 10 shows the distributions of TLP accuracy of different classifiers in 80-20. AW outperforms other classifiers. It is worth noting that AW has been configured to maximize AUC. Therefore, further results in this section refer to the AW classifier only.

Figure 11 shows the TLP accuracy of AW in 80-20.

Table 8 and Table 9 reports the average gain across classifiers in TLP accuracy

Table 7: Statistical test results comparing the accuracy of the same classifier on the same project using moving window over different proximity points. The p-values less than $\alpha = 0.05$ are reported in **Bold**.

Project Name	**Model**	**SMOTE**	**FS**	**PValue(AUC)**	**PValue(F1)**
HBASE	LR	No	No	**0,0001**	**0,0002**
HBASE	LR	No	Yes	**0,0001**	**0,0019**
HBASE	LR	Yes	No	**0,0001**	**0,0001**
HBASE	LR	Yes	Yes	**0,0001**	**0,0023**
HBASE	NN	No	No	**0,0001**	0,1672
HBASE	NN	No	Yes	**0,0001**	0,2568
HBASE	NN	Yes	No	**0,0001**	0,4202
HBASE	NN	Yes	Yes	**0,0001**	0,5011
HBASE	RF	No	No	**0,0001**	**0,0043**
HBASE	RF	No	Yes	**0,0001**	**0,0003**
HBASE	RF	Yes	No	**0,0001**	0,0944
HBASE	RF	Yes	Yes	**0,0001**	**0,0026**
HIVE	LR	No	No	**0,0001**	**0,0433**
HIVE	LR	No	Yes	**0,0001**	**0,0021**
HIVE	LR	Yes	No	**0,0001**	**0,0493**
HIVE	LR	Yes	Yes	**0,0001**	**0,0029**
HIVE	NN	No	No	**0,0001**	0,0682
HIVE	NN	No	Yes	**0,0001**	**0,0180**
HIVE	NN	Yes	No	**0,0001**	**0,0071**
HIVE	NN	Yes	Yes	**0,0001**	**0,0036**
HIVE	RF	No	No	**0,0001**	**0,0042**
HIVE	RF	No	Yes	**0,0001**	**0,0001**
HIVE	RF	Yes	No	**0,0001**	**0,0068**
HIVE	RF	Yes	Yes	**0,0001**	**0,0007**

Table 8: Gain of AW in TLP accuracy using 80-20 in HBASE when increasing the proximity.

	HBASE						
	Precision	**Recall**	**F1**	**AUC**	**GMean**	**Specificity**	**Kappa**
OpenToInProgress	2%	-10%	-3%	19%	74%	240%	149%
InProgressToClosed	21%	5%	15%	22%	29%	58%	117%

using 80-20 when increasing the proximity. These results are similar to the ones obtained in the Sliding Window approach.

As for the sliding-window approach, we expect that decreasing the proximity decreases the accuracy of the predictions, although they increase in practical value since they are available earlier. The interesting point of this research is to evaluate how much and in which direction the accuracy decreases. According to Table 8 and Table 9, we note that:

- The variation differs across metrics. For instance, in HBASE, from Open to InProgress, Recall and F1 even increase. This could be due to the different imbalance of the Open dataset, which is smaller than InProgress since it does not contain tickets without the Developer Assigned date.
- AUC, which is the most reliable metric, increases by about 20% when increasing the proximity in both cases (Open to InProgress and InProgress to Closed) in both projects.
- Kappa is the metric increasing more across metrics, it at least doubles when increasing the proximity in both cases (Open to InProgress and InProgress to Closed) in both projects.

According to the results of SlidingWindow and 80-20 we note that SlidingWindow is overall more accurate than 80-20 in AUC. The AUC of SlidingWindow of Closed is

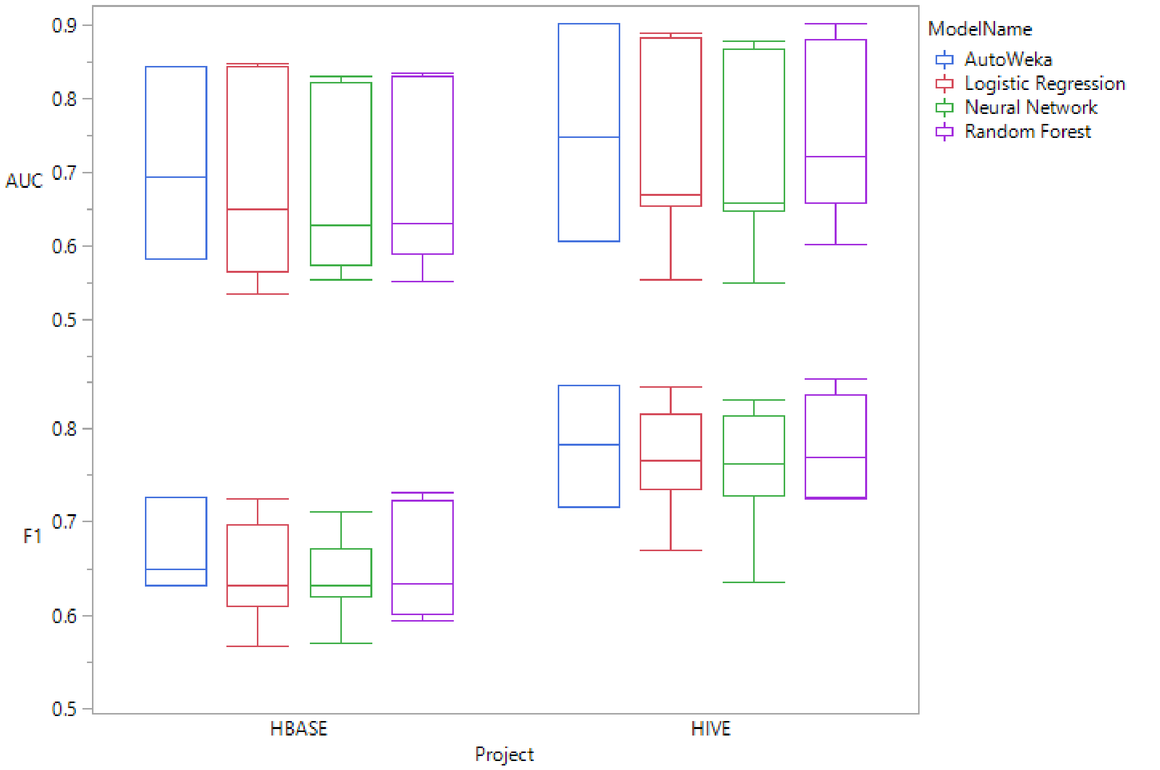


Figure 10: Distributions of TLP accuracy of different classifiers in 80-20.

Table 9: Gain of AW in TLP accuracy using 80-20 in HIVE when increasing the proximity

	HIVE						
	Precision	**Recall**	**F1**	**AUC**	**GMean**	**Specificity**	**Kappa**
OpenToInProgress	13%	4%	9%	23%	37%	82%	299%
InProgressToClosed	18%	-4%	8%	21%	63%	175%	168%

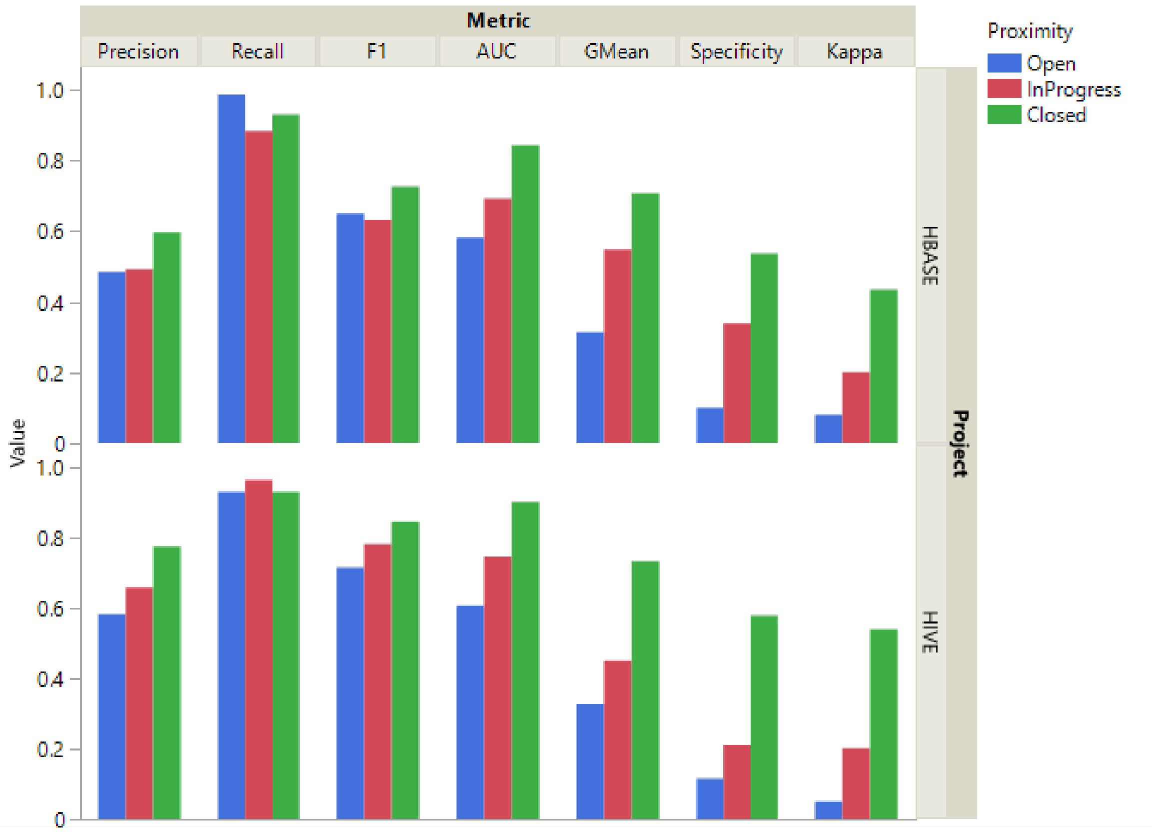


Figure 11: Distributions of TLP accuracy of AW in 80-20.

about 1. This result is consistent with past works and has two main impacts:

- Practitioners should use a sliding window approach given the concept drift in TLP as observed in JIT [72].
- Researchers should focus on the sliding window approach when assessing the power of prediction features.

5.2 Does temporal proximity impact the power of TLP features?

Given that in RQ1 we observed a higher accuracy in sliding window than 80-20, RQ2 results are based on sliding window only.

Figure 12 reports the distributions of feature family power, in terms of IGR, across different proximity points, in specific projects. Figure 13 the distributions of feature family power, in terms of Selected, across different proximity points, in specific projects. According to both figures we note that:

- In Open, the External and Intrinsic feature families have the lowest power;
- Intrinsic has an IGR almost null in all proximity points and both projects despite sometimes it gets selected;
- In closed, the feature families different from JIT decreases their selection proportion but are still selected and their IGR does not change. Therefore, for TLP in Closed, JIT is by far the most important. Feature family and features families different from JIT are still useful for prediction.

Table 10 reports the summary of feature selection results. The entire set of 62 features was subject to feature selection using IGR and counting how many times on average the feature has been selected from every classifier during their evaluation, experimenting on all projects and proximity points. Additionally, we ranked the results of feature selection from the highest to the lowest IGR value. We can see how most of the JIT features occupy the top positions in the ranking, remarkably in conjunction with the Closed proximity point, since it is the stage of the ticket lifecycle where such features are most measurable. If we consider the top 20 features, we start to see some feature from the more ticket-related families. In particular, features related to the number of activities, number of parallel commits, number of different participants and number of different programming languages. Most of ticket-related features present in top 20 have their IGR calculated against the HIVE project, suggesting that such project supply their tickets with more metadata and of better quality than HBASE.

Table 11 reports the Statistical test comparison on the impact on IGR of Feature Family, Proximity and their interaction using moving-window. According to Table 11, we can reject H20 in both projects, and we can claim that the prediction power of features varies according to feature family, proximity point, and their interaction.

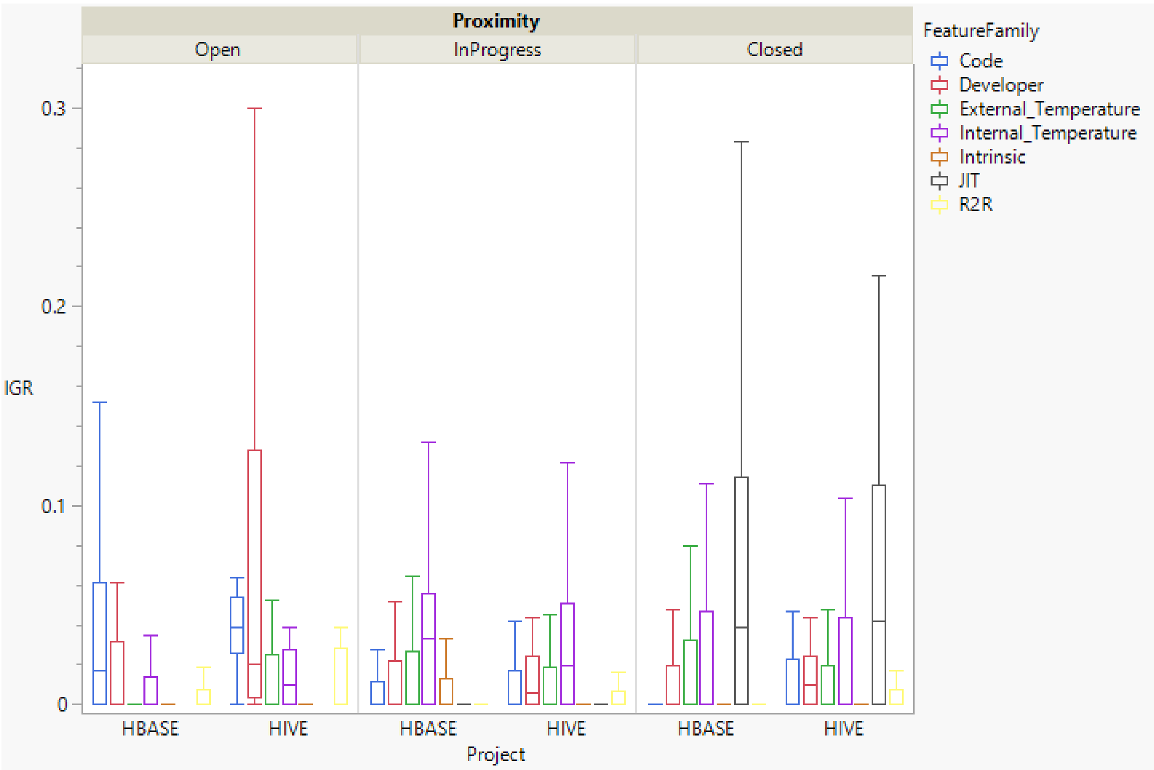


Figure 12: Distributions of feature family power, in terms of IGR, across different proximity points, in specific projects.

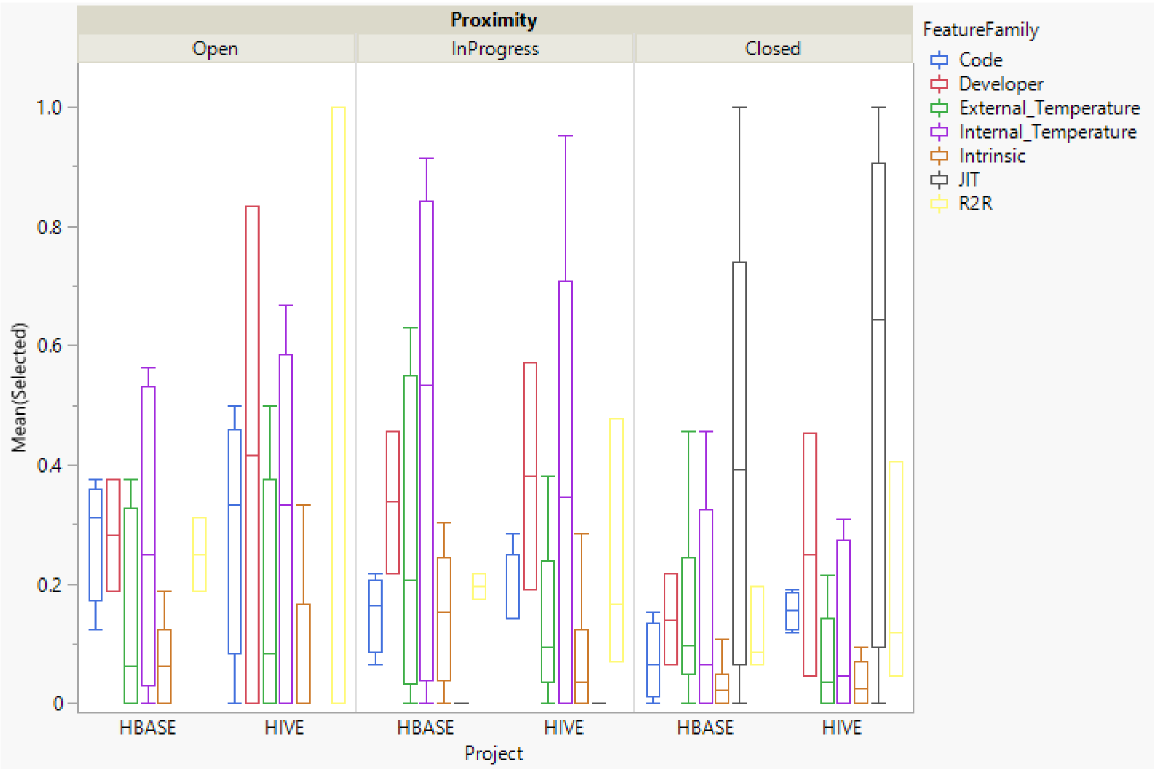


Figure 13: Distributions of feature family power, in terms of Selected, across different proximity points, in specific projects.

Table 10: Summary of feature selection results. The entire set of 62 features was subject to feature selection using IGR (a classifier indipendent metric) and counting how many times on average the metric has been selected for every classifier during their evaluation, experimenting on all projects and proximity points. Additionally, we ranked the results of feature selection from the highest to the lowest IGR value.

Project	**Proximity**	**FeatureFamily**	**Feature**	**Mean(Selected)**	**Mean(IGR)**	**Rank IGR**
HBASE	Closed	JIT	jit-la-SUM	100%	0,23379	1
HBASE	Closed	JIT	jit-nf-MAX	74%	0,16965	2
HIVE	Closed	JIT	jit-la-SUM	100%	0,16054	3
HIVE	Closed	JIT	jit-nf-MAX	93%	0,14653	4
HIVE	Closed	JIT	jit-nd-MAX	90%	0,13980	5
HBASE	Closed	JIT	jit-ent-MAX	85%	0,13900	6
HBASE	Closed	JIT	jit-ld-SUM	98%	0,12490	7
HBASE	Closed	JIT	jit-nd-MAX	65%	0,12443	8
HIVE	Closed	JIT	jit-ent-MAX	64%	0,11848	10
HIVE	Closed	JIT	jit-ld-SUM	100%	0,11457	11
HIVE	Open	External_Temperature	latest_commit-Number_of_files	50%	0,07684	12
HIVE	Open	Internal_Temperature	issue_Participants-count	67%	0,07522	13
HIVE	Closed	JIT	jit-ns-MAX	64%	0,07421	14
HIVE	Open	Code	code_size-number_of_languages	50%	0,07137	15
HBASE	InProgress	Internal_Temperature	activities-count	83%	0,06722	16
HBASE	Closed	Internal_Temperature	activities-count	33%	0,06635	17
HBASE	InProgress	Internal_Temperature	activities-comments_count	91%	0,06476	18
HBASE	Closed	Internal_Temperature	activities-comments_count	46%	0,06339	19
HIVE	InProgress	Internal_Temperature	activities-histories_Count	95%	0,06135	20
HBASE	Closed	JIT	jit-ns-MAX	48%	0,06121	21
HIVE	Closed	Internal_Temperature	activities-histories_Count	69%	0,06010	22
HIVE	Open	Intrinsic	nlp4re_description-EX_ENT	33%	0,06005	23
HIVE	InProgress	Internal_Temperature	nlp4re_sentiment-CM_NNS	67%	0,05979	24
HBASE	Closed	Internal_Temperature	activities-histories_Count	22%	0,05716	25
HIVE	InProgress	Internal_Temperature	activities-count	83%	0,05636	26
HBASE	InProgress	Internal_Temperature	activities-histories_Count	89%	0,05610	27
HIVE	Closed	Internal_Temperature	activities-count	26%	0,05523	28
HBASE	Closed	Internal_Temperature	issue_Participants-count	33%	0,05285	29
HIVE	Open	Internal_Temperature	activities-histories_Count	50%	0,05283	30
HBASE	Closed	JIT	jit-ndev-MAX	39%	0,05271	31
HIVE	InProgress	Intrinsic	type	93%	0,05204	32
HIVE	Open	Intrinsic	type	33%	0,04885	33
HBASE	Closed	External_Temperature	commits_while_in_progress-Churn	46%	0,04798	34
HBASE	InProgress	Internal_Temperature	nlp4re_sentiment-CM_NNS	78%	0,04773	35
HIVE	InProgress	Internal_Temperature	activities-comments_count	52%	0,04755	36
HBASE	InProgress	Internal_Temperature	issue_Participants-count	63%	0,04732	37

Table 10 continued from previous page

Project	Proximity	FeatureFamily	Feature	Mean(Selected)	Mean(IGR)	Rank IGR
HBASE	Closed	JIT	jit-asexp-MIN	7%	0,04544	38
HIVE	Closed	Internal_Temperature	activities-comments_count	24%	0,04503	39
HIVE	Closed	Intrinsic	type	2%	0,04424	40
HIVE	Closed	JIT	jit-fix-COUNT_TRUE	71%	0,04416	41
HIVE	Closed	JIT	jit-asexp-MIN	10%	0,04415	42
HBASE	Open	External_Temperature	temporal_locality	38%	0,04021	44
HIVE	Open	R2R	buggy_similarity-AvgSimilarity_TF-IDF_Cosine_Title	100%	0,03959	45
HBASE	Open	Code	code_quality-Smells_count	31%	0,03952	46
HBASE	Open	Code	code_size-number_of_languages	31%	0,03905	47
HBASE	Open	External_Temperature	temporal_locality-weighted	31%	0,03857	48
HBASE	InProgress	Intrinsic	type	78%	0,03825	49
HIVE	Open	Code	code_size-number_of_files	33%	0,03823	50
HBASE	Closed	JIT	jit-fix-COUNT_TRUE	72%	0,03788	51
HBASE	Closed	Intrinsic	type	2%	0,03786	52
HIVE	InProgress	Internal_Temperature	issue_Participants-count	48%	0,03704	53
HIVE	Closed	Internal_Temperature	issue_Participants-count	31%	0,03695	54
HBASE	Open	Code	code_size-total_LOCs	13%	0,03691	55
HBASE	Closed	External_Temperature	commits_while_in_progress-Count	17%	0,03665	56
HIVE	Open	Code	code_size-total_LOCs	0%	0,03595	57
HIVE	Open	Intrinsic	components-Max_Bugginess	33%	0,03563	58
HBASE	Open	Intrinsic	type	75%	0,03549	59
HBASE	InProgress	Internal_Temperature	nlp4re_sentiment-CM_PNS	43%	0,03312	60
HBASE	InProgress	External_Temperature	commits_while_in_progress-Count	63%	0,03166	61
HBASE	Open	Code	code_size-number_of_files	38%	0,03160	62
HIVE	Open	External_Temperature	temporal_locality	17%	0,03126	63
HIVE	Closed	JIT	jit-ndev-MAX	38%	0,03087	64
HBASE	InProgress	External_Temperature	commits_while_in_progress-Churn	52%	0,02885	65
HIVE	Open	Code	code_quality-Smells_count	33%	0,02664	66
HIVE	Open	Intrinsic	components-Count	17%	0,02563	67
HIVE	InProgress	External_Temperature	commits_while_in_progress-Count	38%	0,02421	68
HIVE	Closed	JIT	jit-age-MIN	71%	0,02396	69
HIVE	Closed	External_Temperature	commits_while_in_progress-Count	12%	0,02379	70
HBASE	Closed	Intrinsic	priority	11%	0,02371	71
HBASE	InProgress	Intrinsic	priority	57%	0,02370	72
HBASE	Closed	JIT	jit-nuc-MAX	7%	0,02357	73
HBASE	InProgress	Internal_Temperature	nlp4re_sentiment-CM_ONS	4%	0,02343	74
HBASE	Closed	Code	code_size-number_of_languages	15%	0,02286	75
HBASE	Open	Internal_Temperature	issue_Participants-count	56%	0,02246	76
HIVE	Closed	External_Temperature	commits_while_in_progress-Churn	21%	0,02244	77
HIVE	Closed	JIT	jit-nuc-MAX	45%	0,02228	78
HIVE	Closed	Intrinsic	priority	38%	0,02121	79
HIVE	InProgress	Intrinsic	priority	62%	0,02080	80

Table 10 continued from previous page

Project	Proximity	FeatureFamily	Feature	Mean(Selected)	Mean(IGR)	Rank IGR
HBASE	Open	Intrinsic	priority	56%	0,02040	81
HBASE	Closed	Developer	assignee-ANFIC	22%	0,02017	82
HBASE	InProgress	Developer	assignee-ANFIC	46%	0,01905	83
HIVE	Closed	Developer	assignee-ANFIC	45%	0,01874	84
HBASE	InProgress	Code	code_size-number_of_languages	15%	0,01862	85
HBASE	InProgress	Internal_Temperature	nlp4re_sentiment-IT_POL	41%	0,01861	86
HBASE	InProgress	Code	code_quality-Smells_count	22%	0,01846	87
HIVE	InProgress	Developer	assignee-ANFIC	57%	0,01823	88
HIVE	InProgress	External_Temperature	commits_while_in_progress-Churn	19%	0,01738	89
HBASE	Closed	JIT	jit-aexp-MIN	9%	0,01679	90
HBASE	Closed	External_Temperature	temporal_locality-weighted	9%	0,01660	91
HIVE	InProgress	Internal_Temperature	nlp4re_sentiment-CM_PNS	21%	0,01650	92
HBASE	InProgress	External_Temperature	temporal_locality-weighted	20%	0,01648	93
HBASE	Open	R2R	buggy_similarity-MaxSimilarity_Levenshtein_Title	31%	0,01622	94
HBASE	Closed	JIT	jit-arexp-MIN	17%	0,01605	95
HIVE	Open	Internal_Temperature	activities-count	33%	0,01590	96
HBASE	Closed	Code	code_quality-Smells_count	9%	0,01579	97
HBASE	InProgress	Code	code_size-total_LOCs	7%	0,01566	98
HBASE	Closed	External_Temperature	temporal_locality	11%	0,01556	99
HBASE	InProgress	External_Temperature	temporal_locality	22%	0,01555	100
HIVE	Open	Intrinsic	priority	0%	0,01465	101
HIVE	Closed	Intrinsic	components-Max_Bugginess	24%	0,01461	102
HIVE	Closed	Code	code_size-number_of_languages	17%	0,01428	103
HIVE	Open	Internal_Temperature	activities-comments_count	0%	0,01412	104
HIVE	InProgress	Code	code_size-number_of_languages	29%	0,01410	105
HIVE	InProgress	Intrinsic	components-Max_Bugginess	29%	0,01397	106
HIVE	Closed	Code	code_size-total_LOCs	19%	0,01350	108
HIVE	Closed	Code	code_size-number_of_files	12%	0,01306	109
HBASE	Closed	Code	code_size-total_LOCs	0%	0,01305	110
HBASE	InProgress	Code	code_size-number_of_files	17%	0,01099	112
HIVE	Closed	Code	code_quality-Smells_count	14%	0,01085	113
HBASE	Closed	Code	code_size-number_of_files	4%	0,01057	114
HBASE	Closed	JIT	jit-age-MIN	24%	0,01055	115
HIVE	InProgress	Internal_Temperature	nlp4re_sentiment-CM_ONS	0%	0,01006	116
HBASE	Open	Internal_Temperature	activities-histories_Count	50%	0,01001	117
HIVE	InProgress	Code	code_size-number_of_files	14%	0,00925	118
HIVE	Open	Intrinsic	nlp4re_description-EX_AMG	17%	0,00922	119
HBASE	InProgress	Intrinsic	nlp4re_description-DA_RKL	7%	0,00917	120
HIVE	Open	Intrinsic	nlp4re_description-DA_IMP	0%	0,00915	121
HBASE	InProgress	Intrinsic	nlp4re_description-EX_RDS	22%	0,00902	122
HBASE	Closed	Developer	assignee-Familiarity	7%	0,00899	123
HIVE	InProgress	Code	code_quality-Smells_count	14%	0,00894	124

Table 10 continued from previous page

Project	Proximity	FeatureFamily	Feature	Mean(Selected)	Mean(IGR)	Rank IGR
HIVE	InProgress	R2R	buggy_similarity-AvgSimilarity_TF-IDF_Cosine_Title	48%	0,00893	125
HIVE	Open	External_Temperature	temporal_locality-weighted	0%	0,00880	126
HBASE	InProgress	Developer	assignee-Familiarity	22%	0,00877	127
HBASE	InProgress	Intrinsic	nlp4re_description-DA_ACT	15%	0,00873	128
HIVE	Closed	Internal_Temperature	nlp4re_sentiment-CM_NNS	7%	0,00845	129
HBASE	InProgress	Intrinsic	nlp4re_description-EX_VRB	0%	0,00835	130
HBASE	InProgress	Intrinsic	nlp4re_description-EX_AMG	30%	0,00833	131
HIVE	Open	Intrinsic	nlp4re_description-EX_ICP	17%	0,00804	132
HBASE	InProgress	Intrinsic	nlp4re_description-EX_CNS	20%	0,00774	133
HIVE	InProgress	Code	code_size-total_LOCs	14%	0,00728	134
HBASE	InProgress	Intrinsic	nlp4re_description-EX_ACD	24%	0,00713	135
HBASE	Open	R2R	buggy_similarity-AvgSimilarity_TF-IDF_Cosine_Title	25%	0,00686	136
HIVE	Closed	R2R	buggy_similarity-AvgSimilarity_TF-IDF_Cosine_Title	40%	0,00683	137
HBASE	InProgress	R2R	buggy_similarity-AvgSimilarity_BagOfWords_Cosine_Text	22%	0,00666	138
HIVE	Closed	Developer	assignee-Familiarity	5%	0,00646	139
HIVE	InProgress	Developer	assignee-Familiarity	19%	0,00642	140
HBASE	Closed	R2R	buggy_similarity-AvgSimilarity_BagOfWords_Cosine_Text	9%	0,00625	141
HBASE	InProgress	Intrinsic	nlp4re_description-DA_IMP	26%	0,00598	142
HBASE	Open	Intrinsic	components-Max_Bugginess	13%	0,00591	143
HIVE	InProgress	External_Temperature	temporal_locality-weighted	12%	0,00588	144
HBASE	InProgress	Intrinsic	nlp4re_description-DA_OPT	15%	0,00582	145
HIVE	Closed	JIT	jit-arexp-MIN	12%	0,00576	146
HBASE	Open	Internal_Temperature	activities-comments_count	25%	0,00573	147
HIVE	Closed	External_Temperature	temporal_locality-weighted	0%	0,00558	148
HBASE	InProgress	R2R	buggy_similarity-AvgSimilarity_TF-IDF_Cosine_Title	17%	0,00551	149
HIVE	Closed	JIT	jit-aexp-MIN	2%	0,00532	150
HBASE	Closed	R2R	buggy_similarity-AvgSimilarity_TF-IDF_Cosine_Title	20%	0,00521	151
HBASE	Open	Intrinsic	components-Count	44%	0,00502	152
HIVE	Open	Intrinsic	nlp4re_description-EX_RDS	33%	0,00488	153
HIVE	Closed	External_Temperature	temporal_locality	0%	0,00481	154
HIVE	InProgress	Intrinsic	nlp4re_description-DA_RKL	5%	0,00477	155
HBASE	Open	Internal_Temperature	activities-count	6%	0,00476	156
HIVE	Open	External_Temperature	commits_while_in_progress-Count	33%	0,00473	157
HIVE	InProgress	External_Temperature	temporal_locality	5%	0,00464	158
HBASE	InProgress	R2R	buggy_similarity-MaxSimilarity_Levenshtein_Title	20%	0,00459	159
HBASE	InProgress	Intrinsic	components-Max_Bugginess	9%	0,00446	160
HIVE	InProgress	Intrinsic	nlp4re_description-EX_VRB	2%	0,00433	161
HBASE	Closed	R2R	buggy_similarity-MaxSimilarity_Levenshtein_Title	7%	0,00430	162
HIVE	Closed	R2R	buggy_similarity-AvgSimilarity_BagOfWords_Cosine_Text	12%	0,00419	163
HBASE	Open	Intrinsic	nlp4re_description-DA_CND	19%	0,00395	164
HBASE	InProgress	Intrinsic	nlp4re_description-EX_SBJ	4%	0,00385	165
HBASE	Closed	Intrinsic	components-Max_Bugginess	2%	0,00377	166

Table 10 continued from previous page

Project	Proximity	FeatureFamily	Feature	Mean(Selected)	Mean(IGR)	Rank IGR
HBASE	InProgress	Intrinsic	components-Count	20%	0,00368	167
HIVE	InProgress	Intrinsic	nlp4re_description-DA_ACT	5%	0,00357	168
HBASE	InProgress	Intrinsic	nlp4re_description-EX_DIR	26%	0,00348	169
HBASE	Closed	External_Temperature	latest_commit-Number_of_files	7%	0,00342	170
HIVE	Closed	R2R	buggy_similarity-MaxSimilarity_Levenshtein_Title	5%	0,00331	171
HBASE	InProgress	Intrinsic	nlp4re_description-DA_CND	11%	0,00322	172
HBASE	Closed	Intrinsic	components-Count	11%	0,00315	173
HIVE	Open	External_Temperature	latest_commit-Churn	0%	0,00301	174
HIVE	InProgress	R2R	buggy_similarity-MaxSimilarity_Levenshtein_Title	17%	0,00296	175
HBASE	Open	R2R	buggy_similarity-AvgSimilarity_BagOfWords_Cosine_Text	19%	0,00295	176
HIVE	InProgress	Internal_Temperature	nlp4re_sentiment-IT_POL	14%	0,00289	177
HBASE	InProgress	External_Temperature	latest_commit-Number_of_files	4%	0,00285	178
HBASE	Open	Intrinsic	nlp4re_description-DA_OPT	13%	0,00280	179
HIVE	InProgress	Intrinsic	nlp4re_description-DA_IMP	14%	0,00278	180
HBASE	InProgress	Intrinsic	nlp4re_description-EX_ICP	15%	0,00269	181
HIVE	InProgress	Intrinsic	nlp4re_description-EX_DIR	17%	0,00263	182
HIVE	Closed	External_Temperature	latest_commit-Number_of_files	7%	0,00258	183
HIVE	InProgress	External_Temperature	latest_commit-Number_of_files	7%	0,00244	184
HBASE	InProgress	Intrinsic	nlp4re_description-EX_ENT	17%	0,00241	185
HBASE	Closed	Internal_Temperature	nlp4re_sentiment-IT_POL	7%	0,00240	186
HBASE	Open	Intrinsic	nlp4re_description-EX_RDS	13%	0,00221	187
HBASE	Open	Intrinsic	nlp4re_description-DA_SRC	13%	0,00213	188
HBASE	Open	Intrinsic	nlp4re_description-EX_AMG	13%	0,00210	189
HBASE	Closed	JIT	num_commits	0%	0,00196	190
HIVE	Open	Intrinsic	nlp4re_description-EX_SBJ	0%	0,00189	191
HIVE	Closed	Intrinsic	nlp4re_description-EX_DIR	10%	0,00180	192
HBASE	Closed	Internal_Temperature	nlp4re_sentiment-CM_NNS	7%	0,00176	193
HIVE	InProgress	Intrinsic	components-Count	12%	0,00167	194
HBASE	Open	External_Temperature	commits_while_in_progress-Churn	13%	0,00166	195
HIVE	Closed	Intrinsic	nlp4re_description-DA_IMP	7%	0,00155	196
HIVE	Closed	Internal_Temperature	nlp4re_sentiment-IT_POL	2%	0,00151	197
HBASE	Closed	JIT	jit-author_date-DURATION	0%	0,00151	198
HBASE	Closed	Intrinsic	nlp4re_description-DA_IMP	7%	0,00151	199
HBASE	Open	External_Temperature	commits_while_in_progress-Count	0%	0,00151	200
HBASE	Closed	Intrinsic	nlp4re_description-DA_OPT	9%	0,00144	201
HIVE	InProgress	Intrinsic	nlp4re_description-DA_CND	7%	0,00139	202
HIVE	InProgress	Intrinsic	nlp4re_description-EX_CNS	0%	0,00136	203
HIVE	InProgress	Intrinsic	nlp4re_description-DA_OPT	5%	0,00133	204
HIVE	InProgress	R2R	buggy_similarity-AvgSimilarity_BagOfWords_Cosine_Text	7%	0,00119	205
HBASE	Closed	Intrinsic	nlp4re_description-DA_CND	4%	0,00114	206
HIVE	Closed	Intrinsic	nlp4re_description-EX_ACD	7%	0,00114	207
HIVE	Closed	Intrinsic	nlp4re_description-EX_RDS	10%	0,00113	208

Table 10 continued from previous page

Project	Proximity	FeatureFamily	Feature	Mean(Selected)	Mean(IGR)	Rank IGR
HBASE	Closed	Intrinsic	nlp4re_description-DA_CNT	9%	0,00100	209
HBASE	Closed	Intrinsic	nlp4re_description-EX_AMG	2%	0,00099	210
HIVE	InProgress	Intrinsic	nlp4re_description-EX_ACD	7%	0,00096	211
HBASE	InProgress	External_Temperature	latest_commit-Churn	0%	0,00092	212
HBASE	Open	Intrinsic	nlp4re_description-EX_CNS	0%	0,00089	213
HBASE	Open	Intrinsic	nlp4re_description-EX_SBJ	0%	0,00083	214
HIVE	Closed	Intrinsic	nlp4re_description-EX_AMG	5%	0,00080	215
HIVE	Closed	Intrinsic	nlp4re_description-DA_ACT	2%	0,00080	216
HIVE	Closed	Intrinsic	nlp4re_description-EX_VRB	0%	0,00079	217
HBASE	Open	Intrinsic	nlp4re_description-DA_ACT	6%	0,00077	218
HIVE	InProgress	Intrinsic	nlp4re_description-EX_RDS	2%	0,00074	219
HIVE	Closed	Intrinsic	components-Count	5%	0,00074	220
HBASE	Open	Intrinsic	nlp4re_description-DA_WKP	13%	0,00071	221
HIVE	Closed	Internal_Temperature	nlp4re_sentiment-CM_ONS	0%	0,00070	222
HIVE	Closed	Intrinsic	nlp4re_description-EX_ENT	5%	0,00064	223
HIVE	Closed	Intrinsic	nlp4re_description-DA_CND	2%	0,00056	224
HBASE	Closed	Intrinsic	nlp4re_description-EX_ENT	4%	0,00051	225
HBASE	Open	Intrinsic	nlp4re_description-DA_CNT	6%	0,00049	226
HIVE	InProgress	Intrinsic	nlp4re_description-EX_ICP	0%	0,00049	227
HBASE	InProgress	Intrinsic	nlp4re_description-DA_WKP	2%	0,00033	228
HIVE	InProgress	Intrinsic	nlp4re_description-EX_SBJ	0%	0,00031	229
HIVE	InProgress	Internal_Temperature	nlp4re_sentiment-IT_SUB	0%	0,00031	230
HBASE	InProgress	Intrinsic	nlp4re_description-DA_CNT	0%	0,00029	231
HBASE	InProgress	Intrinsic	nlp4re_description-DA_SRC	2%	0,00027	232
HIVE	Closed	Internal_Temperature	nlp4re_sentiment-IT_SUB	2%	0,00027	233
HBASE	Closed	Intrinsic	nlp4re_description-EX_VRB	0%	0,00025	234
HBASE	InProgress	Internal_Temperature	nlp4re_sentiment-IT_SUB	2%	0,00024	235
HBASE	Closed	Intrinsic	nlp4re_description-DA_SRC	2%	0,00022	236
HBASE	Closed	External_Temperature	latest_commit-Churn	0%	0,00021	237
HIVE	Closed	Internal_Temperature	nlp4re_sentiment-CM_PNS	0%	0,00018	238
HIVE	Closed	Intrinsic	nlp4re_description-DA_WKP	2%	0,00018	239
HBASE	Closed	Intrinsic	nlp4re_description-DA_WKP	2%	0,00017	240
HBASE	InProgress	Intrinsic	nlp4re_description-DA_INC	0%	0,00014	241
HIVE	Closed	Intrinsic	nlp4re_description-EX_ICP	0%	0,00013	242
HBASE	Closed	Intrinsic	nlp4re_description-DA_INC	0%	0,00011	243
HIVE	Closed	Intrinsic	nlp4re_description-DA_INC	0%	0,00008	244
HBASE	Open	External_Temperature	latest_commit-Churn	0%	0,00000	245
HBASE	Open	External_Temperature	latest_commit-Number_of_files	0%	0,00000	246
HBASE	Open	Internal_Temperature	activities-work_items_Count	0%	0,00000	247
HBASE	Open	Intrinsic	nlp4re_description-DA_IMP	0%	0,00000	248
HBASE	Open	Intrinsic	nlp4re_description-DA_INC	0%	0,00000	249
HBASE	Open	Intrinsic	nlp4re_description-DA_RKL	0%	0,00000	250

Table 10 continued from previous page

Project	Proximity	FeatureFamily	Feature	Mean(Selected)	Mean(IGR)	Rank IGR
HBASE	Open	Intrinsic	nlp4re_description-EX_ACD	0%	0,00000	251
HBASE	Open	Intrinsic	nlp4re_description-EX_DIR	0%	0,00000	252
HBASE	Open	Intrinsic	nlp4re_description-EX_ENT	0%	0,00000	253
HBASE	Open	Intrinsic	nlp4re_description-EX_ICP	0%	0,00000	254
HBASE	Open	Intrinsic	nlp4re_description-EX_VRB	0%	0,00000	255
HBASE	InProgress	Internal_Temperature	activities-work_items_Count	0%	0,00000	256
HBASE	InProgress	JIT	num_commits	0%	0,00000	257
HBASE	Closed	Internal_Temperature	activities-work_items_Count	0%	0,00000	258
HBASE	Closed	Internal_Temperature	nlp4re_sentiment-CM_ONS	0%	0,00000	259
HBASE	Closed	Internal_Temperature	nlp4re_sentiment-CM_PNS	0%	0,00000	260
HBASE	Closed	Internal_Temperature	nlp4re_sentiment-IT_SUB	0%	0,00000	261
HBASE	Closed	Intrinsic	nlp4re_description-DA_ACT	0%	0,00000	262
HBASE	Closed	Intrinsic	nlp4re_description-DA_RKL	0%	0,00000	263
HBASE	Closed	Intrinsic	nlp4re_description-EX_ACD	0%	0,00000	264
HBASE	Closed	Intrinsic	nlp4re_description-EX_CNS	0%	0,00000	265
HBASE	Closed	Intrinsic	nlp4re_description-EX_DIR	0%	0,00000	266
HBASE	Closed	Intrinsic	nlp4re_description-EX_ICP	0%	0,00000	267
HBASE	Closed	Intrinsic	nlp4re_description-EX_RDS	0%	0,00000	268
HBASE	Closed	Intrinsic	nlp4re_description-EX_SBJ	0%	0,00000	269
HIVE	Open	External_Temperature	commits_while_in_progress-Churn	0%	0,00000	270
HIVE	Open	Internal_Temperature	activities-work_items_Count	0%	0,00000	271
HIVE	Open	Intrinsic	nlp4re_description-DA_ACT	0%	0,00000	272
HIVE	Open	Intrinsic	nlp4re_description-DA_CND	0%	0,00000	273
HIVE	Open	Intrinsic	nlp4re_description-DA_CNT	0%	0,00000	274
HIVE	Open	Intrinsic	nlp4re_description-DA_INC	0%	0,00000	275
HIVE	Open	Intrinsic	nlp4re_description-DA_OPT	0%	0,00000	276
HIVE	Open	Intrinsic	nlp4re_description-DA_RKL	0%	0,00000	277
HIVE	Open	Intrinsic	nlp4re_description-DA_SRC	0%	0,00000	278
HIVE	Open	Intrinsic	nlp4re_description-DA_WKP	0%	0,00000	279
HIVE	Open	Intrinsic	nlp4re_description-EX_ACD	0%	0,00000	280
HIVE	Open	Intrinsic	nlp4re_description-EX_CNS	0%	0,00000	281
HIVE	Open	Intrinsic	nlp4re_description-EX_DIR	0%	0,00000	282
HIVE	Open	Intrinsic	nlp4re_description-EX_VRB	0%	0,00000	283
HIVE	Open	R2R	buggy_similarity-AvgSimilarity_BagOfWords_Cosine_Text	0%	0,00000	284
HIVE	Open	R2R	buggy_similarity-MaxSimilarity_Levenshtein_Title	0%	0,00000	285
HIVE	InProgress	External_Temperature	latest_commit-Churn	0%	0,00000	286
HIVE	InProgress	Internal_Temperature	activities-work_items_Count	0%	0,00000	287
HIVE	InProgress	Intrinsic	nlp4re_description-DA_CNT	0%	0,00000	288
HIVE	InProgress	Intrinsic	nlp4re_description-DA_INC	0%	0,00000	289
HIVE	InProgress	Intrinsic	nlp4re_description-DA_SRC	0%	0,00000	290
HIVE	InProgress	Intrinsic	nlp4re_description-DA_WKP	0%	0,00000	291
HIVE	InProgress	Intrinsic	nlp4re_description-EX_AMG	0%	0,00000	292

Table 10 continued from previous page

Project	Proximity	FeatureFamily	Feature	Mean(Selected)	Mean(IGR)	Rank IGR
HIVE	InProgress	Intrinsic	nlp4re_description-EX_ENT	0%	0,00000	293
HIVE	InProgress	JIT	num_commits	0%	0,00000	294
HIVE	Closed	External_Temperature	latest_commit-Churn	0%	0,00000	295
HIVE	Closed	Internal_Temperature	activities-work_items_Count	0%	0,00000	296
HIVE	Closed	Intrinsic	nlp4re_description-DA_CNT	0%	0,00000	297
HIVE	Closed	Intrinsic	nlp4re_description-DA_OPT	0%	0,00000	298
HIVE	Closed	Intrinsic	nlp4re_description-DA_RKL	0%	0,00000	299
HIVE	Closed	Intrinsic	nlp4re_description-DA_SRC	0%	0,00000	300
HIVE	Closed	Intrinsic	nlp4re_description-EX_CNS	0%	0,00000	301
HIVE	Closed	Intrinsic	nlp4re_description-EX_SBJ	0%	0,00000	302
HIVE	Closed	JIT	jit-author_date-DURATION	0%	0,00000	303
HIVE	Closed	JIT	num_commits	0%	0,00000	304

Table 11: Statistical test comparison on the impact on IGR of Feature Family, Proximity and their interaction using moving-window.

Independent Variable	**Pvalue**	
	HBASE	**HIVE**
FeatureFamily	0.0001	0.0001
Proximity	0.0001	0.0001
Proximity $\times$ FeatureFamily	0.0001	0.0001

5.2.1 Ticket bugginess proportion

We complete the study by analyzing how many tickets are bug inducing in the studied datasets.

Figure 14 and Figure 15 show the proportion of bug-inducing tickets related to a specific type and the frequency of that type, respectively for HBASE and HIVE. In both projects there are few New Feature tickets, but they are mostly bug-inducing, suggesting that the projects have reached a state where new features are hard to add without introducing breaking changes. The Bug type seems to be the most prominent one, and it happens to be the most bug-inducing too.

Figure 16 and Figure 17 show the proportion of bug-inducing tickets related to a specific priority and the frequency of that priority, respectively for HBASE and HIVE. In both projects, the most frequent priority is Major, but the most bug-inducing one is Critical. Besides, the Minor and Trivial priorities are the least bug-inducing, suggesting that low-priority tickets are less likely to be bug-inducing, possibly because working on them is less stressful for the developers.

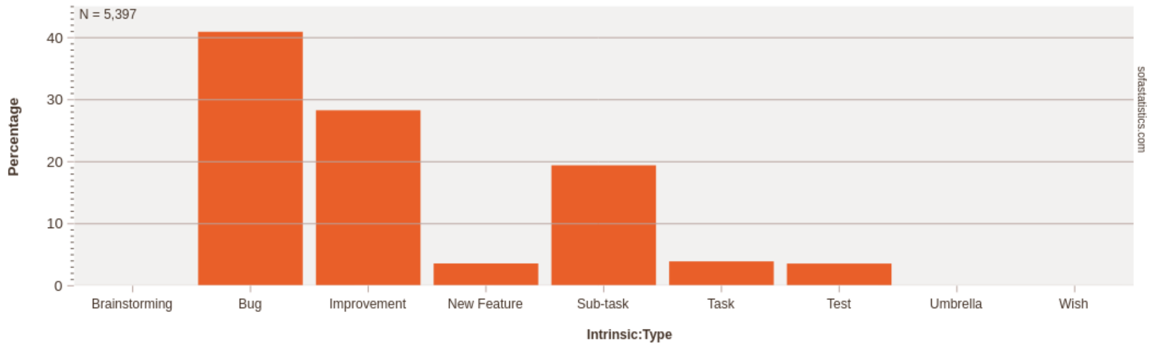
6 Threats to Validity

In this section, we report the threats to the validity of our study. The section is organized by four threat types: Conclusion, Internal, Construct, and External [103].

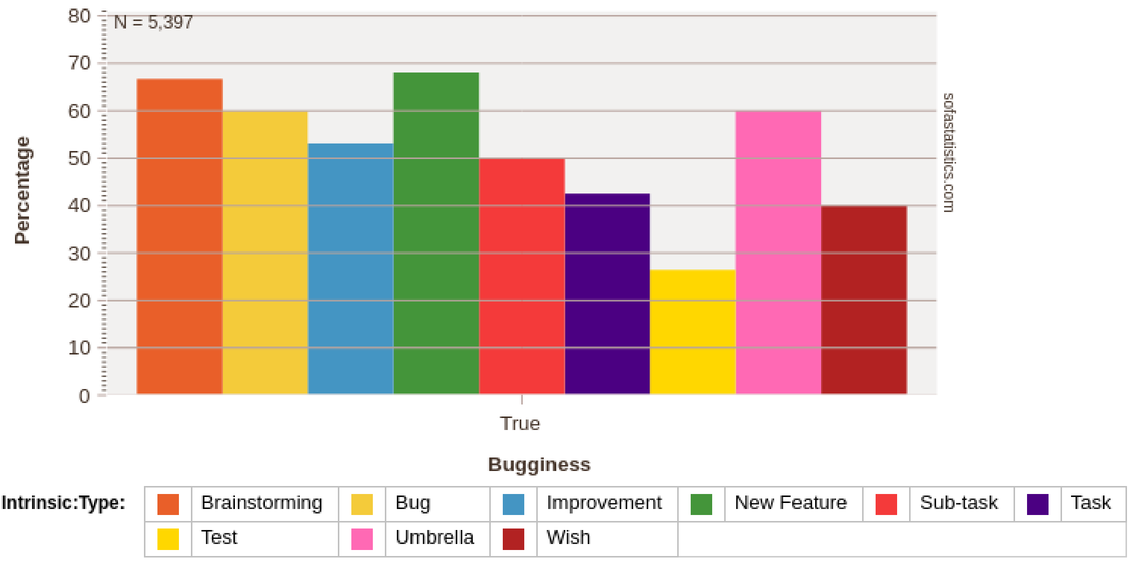
6.1 Conclusion

Conclusion validity addresses factors that influence the accuracy of inferences about the relationship between independent and dependent variables. [103].

A common challenge in software defect prediction research is the gap between reported model accuracy and the demonstrable value these models provide for improving software quality in real-world settings [47]. While this study, like much of the existing literature, concentrates on evaluating the accuracy of our proposed model, we acknowledge that its ultimate value lies in its practical impact. Assessing this impact, however, is beyond the scope of the current research.

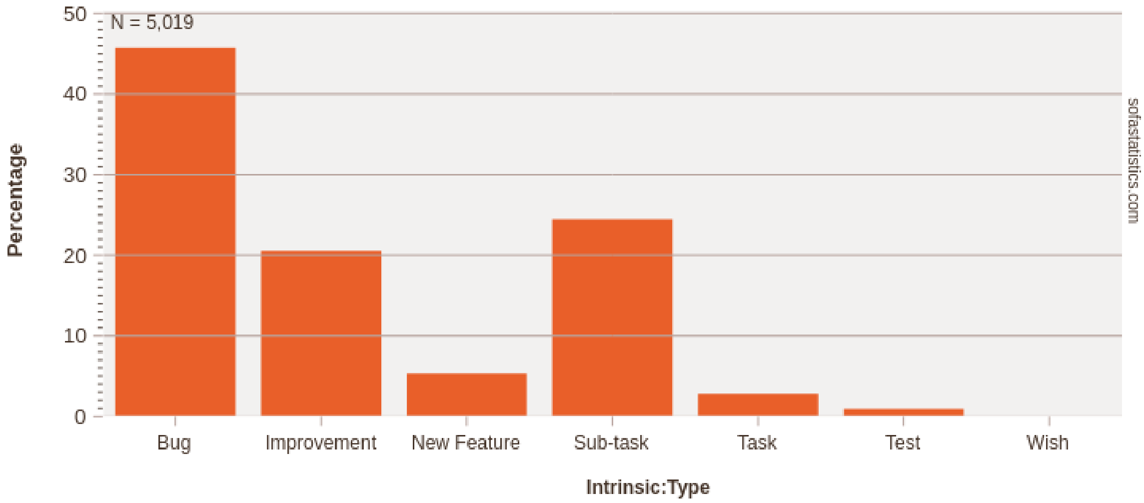


(a) Frequency

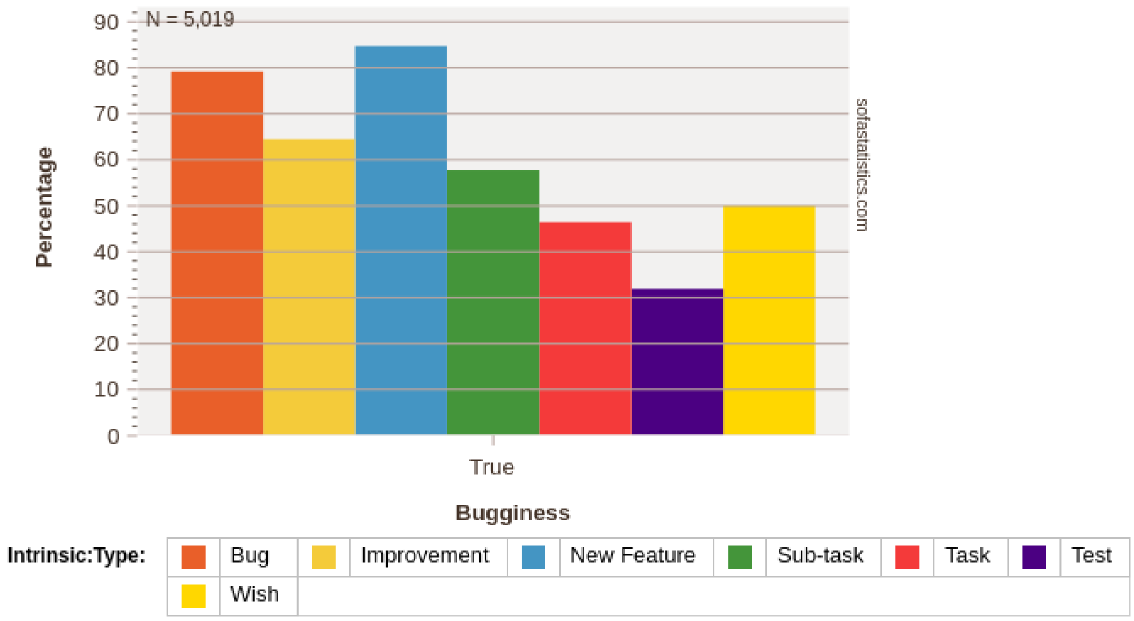


(b) Bug-inducing proportion

Figure 14: Proportion of buggy tickets related to a specific ticket type and the frequency of that type (HBASE).



(a) Frequency



(b) Bug-inducing proportion

Figure 15: Proportion of buggy tickets related to a specific ticket type and the frequency of that type (HIVE).

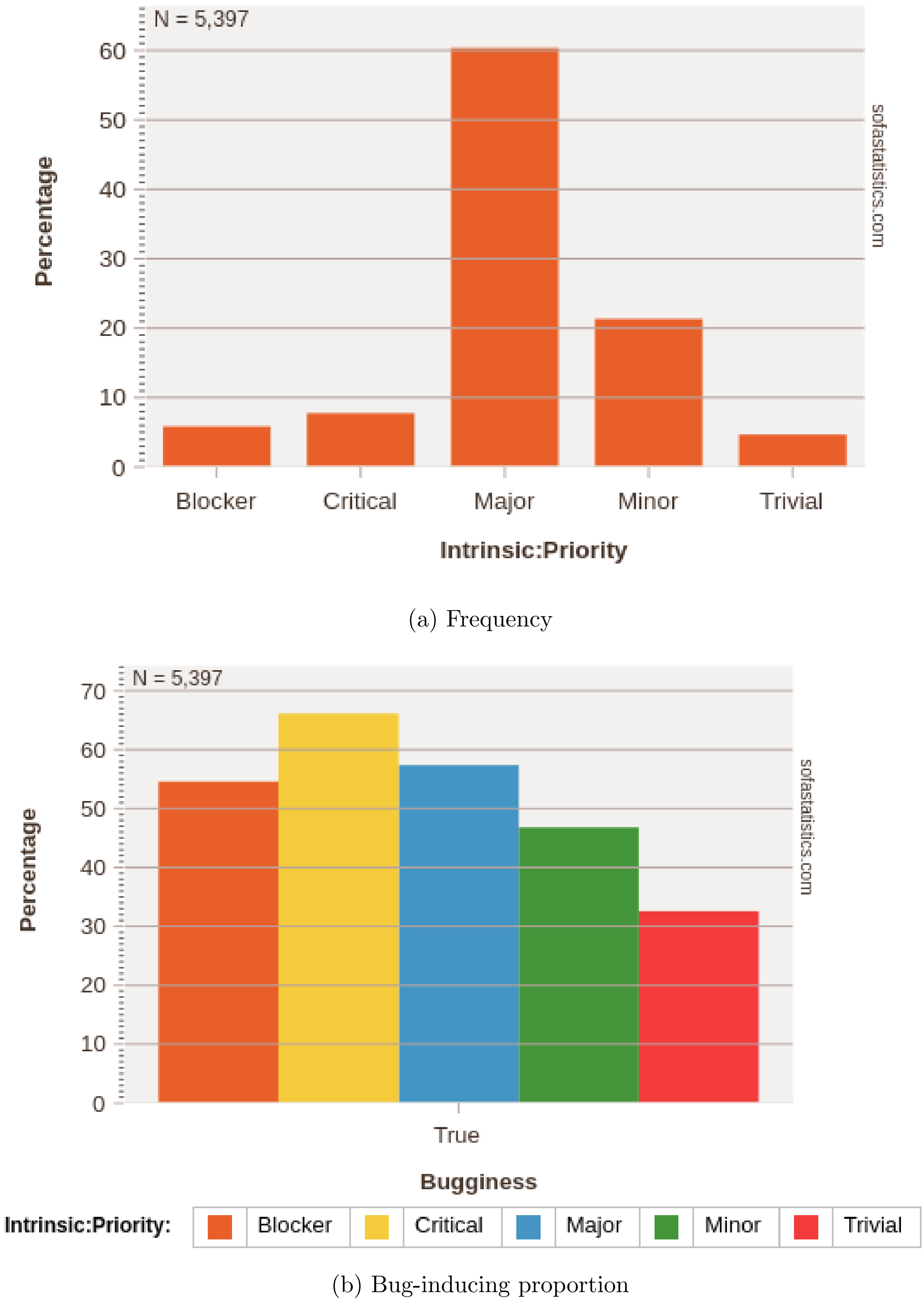
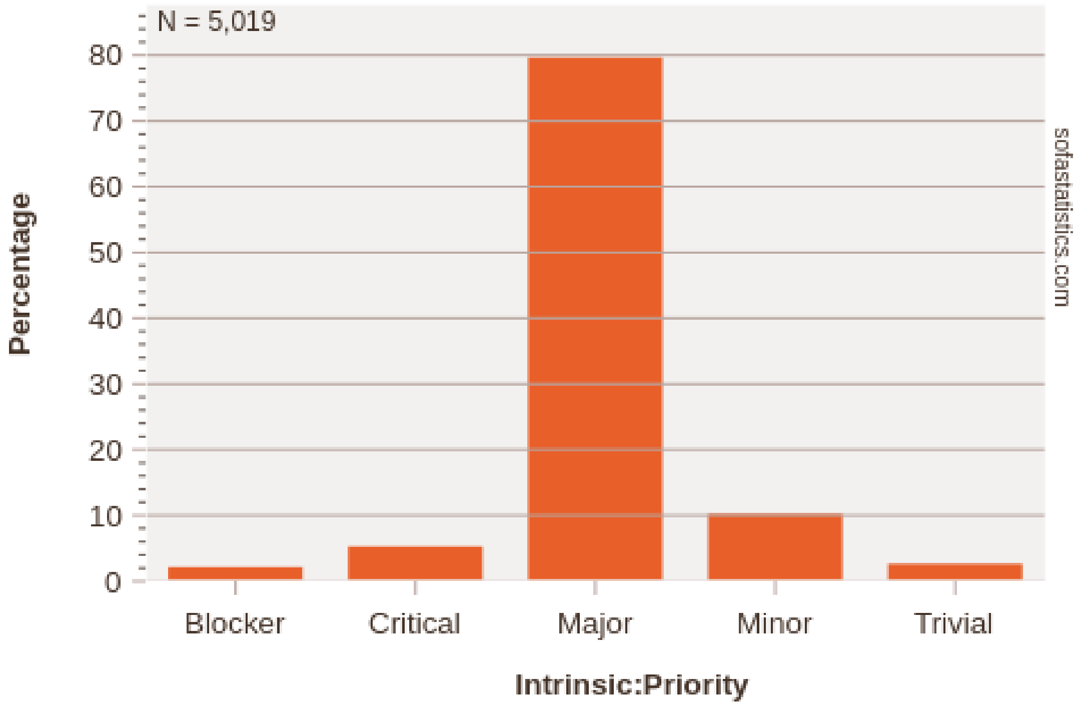
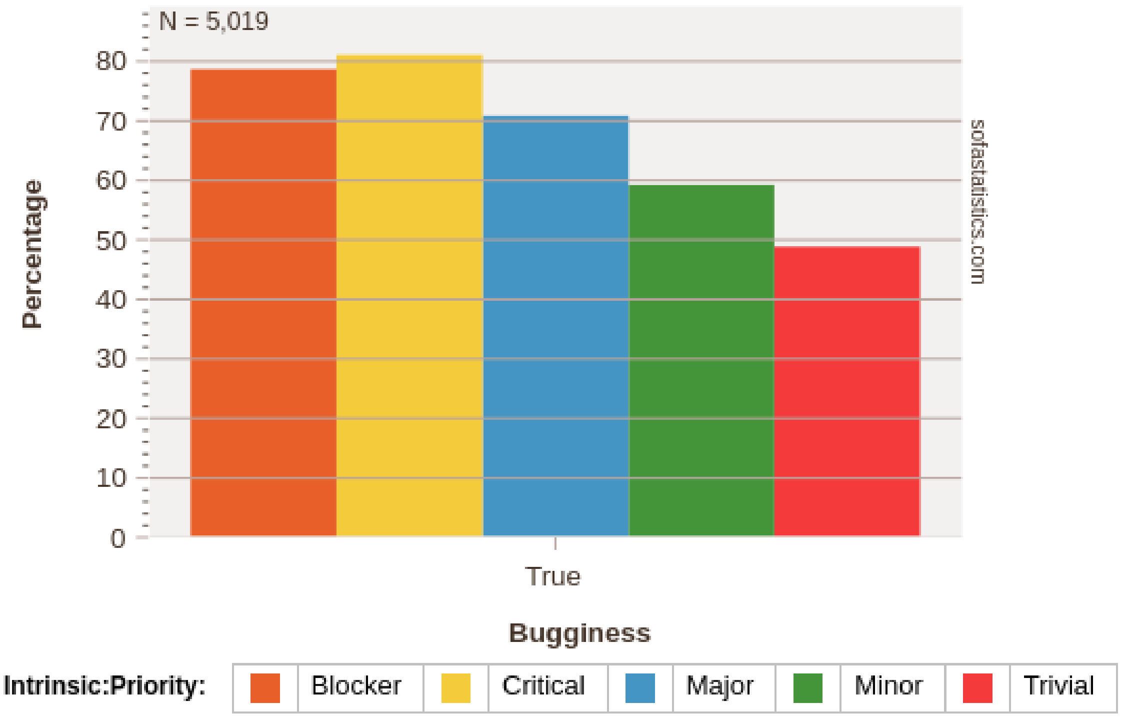


Figure 16: Proportion of buggy tickets related to a specific ticket priority and the frequency of that type (HBASE).



(a) Frequency



(b) Bug-inducing proportion

Figure 17: Proportion of buggy tickets related to a specific ticket priority and the frequency of that type (HIVE).

6.2 Internal

Internal validity addresses elements that could impact the independent variables in establishing causal relationships [103].

Similarly to many other defect prediction studies [91, 28, 92, 90, 36, 23, 95, 77], we do not differentiate between defects based on severity. While defect severity is a relevant factor in practice, there is currently no evidence suggesting that severity significantly impacts defect prediction accuracy results. Therefore, we consider this a potential area for future work.

6.3 Construct

Construct validity is concerned with the degree to which our measurements indeed reflect what we claim to measure [103].

The execution of a prediction study on defect prediction entails many, often subjective, design decisions such as validation technique, balancing, normalization, tuning, and many more, which might influence the prediction results. We do not expect that our design choices coincide with the choices of all readers, our intent is to use state-of-the-art techniques. We have reused many design choices from Patel, Adams, and Hassan [77] including the classifiers and the validation technique. However, we changed some decisions compared to Patel, Adams, and Hassan [77]; for instance, in this paper, we do not balance the data due to the use of AutoWeka, the nature of the unbalance. Since our datasets are unbalanced towards the positives. Moreover, AutoWeka already takes the concept of balancing into consideration when tuning the model. Specifically, Auto-WEKA evaluates a diverse set of classifiers, and some of them are intrinsically robust to class imbalance (e.g., decision trees, random forests). Furthermore, the parameters of some models, such as class weights, can mitigate class imbalance.

AUC was selected as the evaluation metric for feature selection compared to other options like F1 due the intrinsic difficulty in quantifying the specific costs associated with different types of classification errors in the current context; i.e. in the context of ticket prediction, we do not know how much a false positive cost compared to a false negative. Moreover, by prioritizing the relative order of predictions, AUC is better suited than F1 for identifying features that effectively discriminate between classes across different thresholds [48].

In order to avoid dormant defects that would impact our ground-truth [23, 1], we neglected the last 20% of the tickets of each project.

6.4 External

External validity is concerned with the extent to which the research elements (subjects, artifacts, etc.) are representative of actual elements [103].

This study utilized a large set of datasets, comprising approximately 11,000 tickets from two open-source Apache projects. While this dataset is diverse, it may not fully capture the characteristics of proprietary, large-scale, or domain-specific software projects. The generalizability of our findings may be affected by differences in development practices, team structures, and project governance models across various software ecosystems.

Additionally, the ticket granularity and labeling process may vary across different projects, potentially impacting the effectiveness of TLP models when applied to other repositories. The distribution of defect-inducing tickets may also differ based on factors such as project maturity, contributor experience, and issue-tracking conventions.

Another potential threat is the impact of evolving software engineering practices over time. Since our dataset captures historical defect patterns, shifts in coding standards, tooling, or software development methodologies (e.g., DevOps, continuous integration) may alter the relevance of certain features in future software projects.

To mitigate these threats, future work should replicate this study across a broader range of projects, including industrial datasets and repositories from different domains (e.g., financial, healthcare, embedded systems). Additionally, longitudinal studies could help assess the stability of TLP models over time and their adaptability to evolving software engineering environments.

7 Related Work

7.1 Requirements Quality

Requirements quality refers to the degree to which software requirements are well-defined, unambiguous, complete, consistent, and testable [9, 99]. High-quality requirements are essential for guiding development teams and ensuring the final product aligns with stakeholders' needs [97].

Berry and Lawrence [9] emphasize that clear and precise requirements minimise misunderstandings during development, leading to more efficient project execution, and several studies [3, 51, 10] report on the disruptive effects that ambiguity, inconsistency, incompleteness or, more generally, "requirements smells" [38] can have on SW project success [56].

But how can we minimise the effects of requirements smells? The problem can be tackled in two ways: either by avoiding the creation of poor-quality requirements from the beginning or by accelerating the requirements' elicitation process while ensuring quality. Applying standards like INCOSE [52] or ISO 24981 [53] has proven effective in minimizing the occurrence of requirement smells by construction. On the other hand, automated tools, such as those proposed by [30] and [32], have shown to be valuable in detecting smells, by means of using Natural Language Processing (NLP) techniques.

Both approaches rely on the adoption of a structured natural language, constraining the syntax of the requirements, and thus limiting the interpretive flexibility of the person writing them.

But what can we do for projects that do not follow standards when writing requirements? And how should we proceed when no "proper" requirements are available?

While we support these approaches and acknowledge their benefits, we found significant challenges in applying automated tools—primarily designed to detect deviations from standard-based patterns—to open-source projects. The main limitation is the absence of formal software requirements. These projects are typically managed through Issue Tracking Systems (ITS) like Atlassian Jira [67], where the most actionable artefact is a ticket. However, Jira tickets usually describe tasks, bugs, or feature requests in an informal and needs-oriented or implementation-driven manner rather than providing a structured and well-defined specification (as imposed by standards).

From this perspective, to prevent the accumulation of thousands of requirement smells as per detected by these tools, we preferred to focus on identifying broader features to assess the quality of tickets, mimicking the approach of proposed by [17, 99]. These include factors such as the number of actions to be performed, as well as the sentence syntactic completeness, the occurrence of ambiguous words or the presence of weak phrases. More general features, of course, are less actionable in terms of correcting smells, but considering our purpose of predicting defects at Ticket Level, it is reasonable to think that they can bring more valuable information with respect to the "flat" information (in terms of detected smells) that will derive from the application of cited automated tools since they are likely to detect all the smells for each ticket. The details of specific features applied to tickets will be discussed later in section 3 of this paper.

7.2 Developers' Skills

In software development, the actions and decisions of developers play a crucial role in determining the quality and reliability of a system. Faults introduced during development can stem from various factors, including unclear code ownership and lack of responsibility assignment, challenges in distributed collaboration, along with practitioner's specific skills (or lack of them) [8] and (wrong) behaviors [63].

Particularly interesting is the approach adopted by Matsumoto et al. [71], who introduced developer metrics (code churn, number of commits, number of modified files per developer) as a feature for fault prediction, focusing on individual contributions to the software project. We share their approach, and adopted features like the number of different developers involved in project development, the number of authors of comments and attachments for a certain ticket, the proportion of tickets assigned to a particular developer, the mean number of buggy tickets assigned to a particular developer and the number of developers assigned to a specific SW module, to measure the impact of the developers on code implementation.

7.3 Impact Analysis

Change Impact Analysis (CIA) is a critical aspect of software engineering, helping teams assess the consequences of modifications to software systems. By identifying which components are affected by a proposed change, teams can minimize unintended consequences, optimize their development efforts, and establish maintenance and testing strategies [64, 11, 5]. Several studies have explored different dimensions of CIA [37], ranging from classification frameworks to practical applications in software maintenance, automated traceability [6], and prioritization techniques. From this perspective, we concentrated on the Ticket to Code relationship, as retrieved by commit analysis, introducing metrics like "SW Component count", defined as the number of SW components impacted by ticket implementation, and "SW Component bugginess", i.e. the percentage of buggy tickets over the total ticket number insisting on the same SW component, to estimate the impact that each ticket have had on code.

7.4 Defect Prediction

As software projects have grown in size and complexity, with more frequent releases and tighter time constraints, software defect prediction has become increasingly critical in software engineering, helping to identify system components likely to contain defects before they manifest in production [57]. In this context, Just-In-Time Software Defect Predictors (JIT SDP) have been extensively studied as a promising approach to assessing commit quality and predicting potential defects at commit time [107]. While these models have demonstrated their feasibility in enhancing software quality, there is still room for improvement [107]. Their effectiveness depends on several key factors, including the quality of the training dataset [68, 89], parameters optimization [35, 91], retraining frequency [23, 72], and timing [86].

Several studies have proposed the integration of new metrics for enhancing the precision of these models [50, 7], and new features, based on ITS historical data, targeting the granularity of defects at different levels (commit-level, class-level and method-level)[25]. Moreover, a systematic mapping study conducted by Ozakinci and Tarhan [76] showed that most effective early-stage software defect prediction (E-SDP) methods extend beyond traditional code metrics, incorporating the quality of requirements and design artefacts as key predictive factors.

From this perspective, we build upon the previous works by:

- Creating a Ticket Level Prediction dataset derived from (manually cured) JIT dataset
- Applying traditional metrics coming from static code analysis tools
- Adapting reasonable (and introducing new) metrics from requirements analysis, including ITS metadata

- Training and studying the performance of three different models (mimicking the approach of Falessi et al. [28]) with respect to JIT models
- Evaluating the prediction "power" of these models, at three different ticket-lifetime instants

Details on the overall applied methodology and features are reported in section 4 and section 3.

7.5 Temporal Proximity in Predictions

Models prediction accuracy is impacted by time [12, 88, 13, 75], and it is reasonable to assume that the closer we get to the prediction instant, the more information we gain, and the more precise the prediction becomes. So, the concept of temporal proximity in prediction comes to play a significative role, as prediction accuracy typically declines over longer time horizons due to the error accumulation of long-term predictors [12, 75], especially when applied to intrinsically stochastic processes, like weather forecasting [13, 70], financial stock market [69], or epidemic modelling [59].

Models such as Autoregressive Moving Average (ARIMA) [84] and Long Short-Term Memory (LSTM) [43] address this issue by assigning greater weight to, i.e. prioritising, recent observations over long-term ones, allowing short-term factors to have a stronger influence on predictions. Empirical studies [33, 44, 14, 31] have largely demonstrated the benefits of adopting this strategy by the significant precision improvement of the short-term-prediction.

We share these findings and, following the proposed approach, we compared the "predictive power" of three different models over three different time windows, as reported in Figure Figure 3 (at ticket creation time, at ticket assignment time, and at commit time - like JIT SDP do), with the intent of determining which factors, i.e. features, become more important for prediction precision as we approximate to defect-introduction instant.

7.6 NLP on tickets

With the advent of Issue Tracking Systems (IST) like Jira, software development teams generate an overwhelming amount of data in the form of tickets, which document various activities such as bug reports, new feature requests, and updates. Triagers—who are often also developers—must manually process these tickets to properly categorize them, assign them to the right team members, prioritize their resolution [87], other than fix broken (or missing) traceability or identify similar bug reports (clone detection) [24]. This process is not only time-consuming but also prone to human error, and have an extremely high impact on the software maintenance. To address these challenges, researchers have explored different aspects of IST artefacts, particularly focusing on automating the extraction of useful information from tickets using Natural Language Processing (NLP), machine learning, and deep learning.

Their efforts have targeted critical problems such as ticket classification (e.g., distinguishing between bug reports, feature requests, and updates) [4, 81, 108, 27] and resolution prioritization [2, 93, 42, 83], severity level and resolution effort estimation [19], assignment to developers with the appropriate expertise [106], and the detection of duplicate reports [24].

Several works have specifically investigated the role of textual similarity in predicting software defects. Papers [24] and [26] demonstrate that requirements historically associated with changes to a specific class often exhibit semantic similarity to new requirements impacting the same class. Following this principle, we apply NLP techniques to Jira feature request tickets, treating them as if they were requirements. By leveraging textual similarity measures, we aim to predict the likelihood that a feature request may induce defects, under the assumption that new tickets highly similar to past bug-inducing ones are more prone to introducing bugs.

Our work builds on these studies by implementing a tailored approach that combines NLP techniques, aggregation methods, and ticket attributes to enhance predictive performance. The features extracted by mean of NLP techniques are reported section 3.

8 Conclusion

In line with the principle that prevention is better than cure, this thesis introduced and evaluated an initial approach for Ticket-Level Prediction (TLP)—a method aimed at identifying tickets that, when implemented, are likely to introduce bugs. Our approach considers three key temporal points in the lifecycle of a ticket: Open, InProgress, and Closed. To facilitate TLP, we defined and measured 62 features spanning multiple domains, including commit-level and class-level defect prediction, requirements quality, natural language processing (NLP), and broader software engineering metrics.

The evaluation of TLP models involved balancing techniques, feature selection methods, and various machine learning classifiers for bug prediction, applied to approximately 11,000 tickets from two Apache open-source projects. We assessed model performance using Precision, Recall, F1-score, AUC, Kappa, and GMean as accuracy metrics, while information gain ratio and backward search feature selection were used to measure feature importance.

Our results confirm that TLP accuracy improves as proximity to the defect introduction event increases. Notably, the Sliding Window approach outperformed the traditional 80-20 split in terms of AUC, reinforcing prior findings on concept drift in just-in-time (JIT) defect prediction [72]. Consequently, practitioners should favor a sliding window strategy when implementing TLP, while researchers should prioritize it when evaluating predictive features.

Regarding the predictive power of feature families, our analysis reveals that their effectiveness depends on proximity and interactions among them. No single feature family consistently outperforms others across all proximity points. Instead, prediction models should dynamically adapt feature selection based on the proximity stage. In particular, at the Closed stage, JIT-related features become dominant, though other feature families remain relevant, as their selection proportion decreases without significant changes in their information gain ratio. This finding underscores the necessity of leveraging JIT-based features for late-stage predictions, while maintaining a broader set of features for earlier stages.

In conclusion, preventing rather than curing defects can significantly enhance software reliability and reduce testing costs. Moreover, fostering bug prevention techniques can also contribute to software engineering education by instilling practices that help developers minimize defect introduction from the outset.

In the future we plan to extend this work by performing the following investigations:

- Expanding the dataset scope
  - Extend the study to additional open-source and industrial projects to assess generalizability.
  - Investigate how project characteristics (e.g., team size, development methodology) affect TLP performance.
- Refining TLP feature engineering
  - Explore additional feature families, including developer activity metrics and repository evolution trends.
  - Incorporate deep learning-based feature extraction from textual and code-related artifacts.
  - Investigate the role of domain-specific language models in improving NLP-based TLP features.
- Enhancing TLP prediction models
  - Evaluate the impact of ensemble learning approaches on TLP accuracy.
  - Investigate hybrid models combining traditional machine learning with deep learning architectures.
  - Adapt models for real-time TLP integration within software development pipelines.
- Addressing concept drift in TLP
  - Further study the impact of concept drift on prediction stability over time.
  - Develop adaptive learning techniques to dynamically update models as new data arrives.

  - Compare the effectiveness of different sliding window configurations for handling drift.

- Optimizing TLP for software engineering practice
  - Design actionable recommendations for developers based on TLP predictions.
  - Integrate TLP with issue tracking systems and continuous integration pipelines.
  - Evaluate the cost-benefit trade-offs of TLP in real-world software development settings.

- Bridging TLP with software education
  - Investigate how TLP insights can be leveraged to teach better bug prevention strategies.
  - Develop educational tools that incorporate TLP feedback into programming courses.
  - Assess the effectiveness of TLP-driven learning interventions on software quality.

References

[1] Aalok Ahluwalia, Davide Falessi, and Massimiliano Di Penta. "Snoring: a noise in defect prediction datasets". In: *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*. Ed. by Margaret-Anne D. Storey, Bram Adams, and Sonia Haiduc. IEEE / ACM, 2019, pp. 63–67. DOI: 10.1109/MSR.2019.00019. URL: https://doi.org/10.1109/MSR.2019.00019.

[2] Hafiza Anisa Ahmed, Narmeen Zakaria Bawany, and Jawwad Ahmed Shamsi. "CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms". In: *IEEE Access* 9 (2021), pp. 50496–50512. DOI: 10.1109/ACCESS.2021.3069248. URL: https://doi.org/10.1109/ACCESS.2021.3069248.

[3] Jarmo J. Ahonen and Paula Savolainen. "Software engineering projects may fail before they are started: Post-mortem analysis of five cancelled projects". In: *J. Syst. Softw.* 83.11 (2010), pp. 2175–2187. DOI: 10.1016/J.JSS.2010.06.023. URL: https://doi.org/10.1016/j.jss.2010.06.023.

[4] Giuliano Antoniol et al. “Is it a bug or an enhancement?: a text-based approach to classify change requests”. In: *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering, CASCON 2018, Markham, Ontario, Canada, October 29-31, 2018*. Ed. by Iosif-Viorel Onut et al. ACM, 2018, pp. 2–16. URL: https://dl.acm.org/citation.cfm?id=3291293.

[5] Sajid Anwer et al. “Comparative Analysis of Requirement Change Management Challenges Between in-House and Global Software Development: Findings of Literature and Industry Survey”. In: *IEEE Access* 7 (2019), pp. 116585–116611. DOI: 10.1109/ACCESS.2019.2936664. URL: https://doi.org/10.1109/ACCESS.2019.2936664.

[6] Thazin Win Win Aung, Huan Huo, and Yulei Sui. “A Literature Review of Automatic Traceability Links Recovery for Software Change Impact Analysis”. In: *ICPC ’20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*. ACM, 2020, pp. 14–24. DOI: 10.1145/3387904.3389251. URL: https://doi.org/10.1145/3387904.3389251.

[7] Alberto Bacchelli, Marco D’Ambros, and Michele Lanza. “Are Popular Classes More Defect Prone?” In: *Fundamental Approaches to Software Engineering, 13th International Conference, FASE 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*. Ed. by David S. Rosenblum and Gabriele Taentzer. Vol. 6013. Lecture Notes in Computer Science. Springer, 2010, pp. 59–73. DOI: 10.1007/978-3-642-12029-9_5. URL: https://doi.org/10.1007/978-3-642-12029-9%5C_5.

[8] Gunnar R. Bergersen, Dag I. K. Sjøberg, and Tore Dybå. “Construction and Validation of an Instrument for Measuring Programming Skill”. In: *IEEE Trans. Software Eng.* 40.12 (2014), pp. 1163–1184. DOI: 10.1109/TSE.2014.2348997. URL: https://doi.org/10.1109/TSE.2014.2348997.

[9] Daniel M Berry and Brian Lawrence. “Requirements engineering”. In: *IEEE software* 15.2 (1998), pp. 26–29.

[10] Barry Boehm and Victor R Basili. “Software defect reduction top 10 list”. In: *Software engineering: Barry W. Boehm’s lifetime contributions to software development, management, and research* 34.1 (2007), p. 75.

[11] Andréa Sabedra Bordin and Fabiane Barreto Vavassori Benitti. “Software maintenance: what do we teach and what does the industry practice?” In: *Proceedings of the XXXII Brazilian Symposium on Software Engineering, SBES 2018, Sao Carlos, Brazil, September 17-21, 2018*. Ed. by Uirá Kulesza. ACM, 2018, pp. 270–279. DOI: 10.1145/3266237.3266251. URL: https://doi.org/10.1145/3266237.3266251.

[12] George EP Box et al. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[13] Peter J Brockwell and Richard A Davis. *Introduction to time series and forecasting.* Springer, 2002.

[14] Lawrence David Brown. *On the Admissibility of Invariant Estimators of Location Parameters.* Cornell University, 1964.

[15] George G. Cabral et al. "An investigation of online and offline learning models for online Just-in-Time Software Defect Prediction". In: *Empir. Softw. Eng.* 28.5 (2023), p. 121. DOI: 10.1007/S10664-023-10335-6. URL: https://doi.org/10.1007/s10664-023-10335-6.

[16] Aloisio S. Cairo, Glauco de Figueiredo Carneiro, and Miguel P. Monteiro. "The Impact of Code Smells on Software Bugs: A Systematic Literature Review". In: *Inf.* 9.11 (2018), p. 273. DOI: 10.3390/INFO9110273. URL: https://doi.org/10.3390/info9110273.

[17] Nathan Carlson and Phillip A. Laplante. "The NASA automated requirements measurement tool: a reconstruction". In: *Innov. Syst. Softw. Eng.* 10.2 (2014), pp. 77–91. DOI: 10.1007/S11334-013-0225-8. URL: https://doi.org/10.1007/s11334-013-0225-8.

[18] Nitesh V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *J. Artif. Intell. Res.* 16 (2002), pp. 321–357. DOI: 10.1613/JAIR.953. URL: https://doi.org/10.1613/jair.953.

[19] Morakot Choetkiertikul et al. "A Deep Learning Model for Estimating Story Points". In: *IEEE Trans. Software Eng.* 45.7 (2019), pp. 637–656. DOI: 10.1109/TSE.2018.2792473. URL: https://doi.org/10.1109/TSE.2018.2792473.

[20] Tom Copeland. *PMD applied.* Vol. 10. 2005.

[21] Adolfo Crespo Márquez. "The curse of dimensionality". In: *Digital Maintenance Management: Guiding Digital Transformation in Maintenance.* Springer, 2022, pp. 67–86.

[22] Marco D'Ambros, Michele Lanza, and Romain Robbes. "An extensive comparison of bug prediction approaches". In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010).* 2010, pp. 31–41. DOI: 10.1109/MSR.2010.5463279.

[23] Davide Falessi, Aalok Ahluwalia, and Massimiliano Di Penta. "The Impact of Dormant Defects on Defect Prediction: A Study of 19 Apache Projects". In: *ACM Trans. Softw. Eng. Methodol.* 31.1 (2022), 4:1–4:26. DOI: 10.1145/3467895. URL: https://doi.org/10.1145/3467895.

[24] Davide Falessi, Giovanni Cantone, and Gerardo Canfora. “Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques”. In: *IEEE Trans. Software Eng.* 39.1 (2013), pp. 18–44. DOI: 10.1109/TSE.2011.122. URL: https://doi.org/10.1109/TSE.2011.122.

[25] Davide Falessi et al. “Enhancing the defectiveness prediction of methods and classes via JIT”. In: *Empir. Softw. Eng.* 28.2 (2023), p. 37. DOI: 10.1007/S10664-022-10261-Z. URL: https://doi.org/10.1007/s10664-022-10261-z.

[26] Davide Falessi et al. “Leveraging Historical Associations between Requirements and Source Code to Identify Impacted Classes”. In: *IEEE Trans. Software Eng.* 46.4 (2020), pp. 420–441. DOI: 10.1109/TSE.2018.2861735. URL: https://doi.org/10.1109/TSE.2018.2861735.

[27] Davide Falessi et al. “On failure classification: the impact of ”getting it wrong””. In: *36th International Conference on Software Engineering, ICSE ’14, Companion Proceedings, Hyderabad, India, May 31 - June 07, 2014*. Ed. by Pankaj Jalote, Lionel C. Briand, and André van der Hoek. ACM, 2014, pp. 512–515. DOI: 10.1145/2591062.2591122. URL: https://doi.org/10.1145/2591062.2591122.

[28] Davide Falessi et al. “On the need of preserving order of data when validating within-project defect classifiers”. In: *Empir. Softw. Eng.* 25.6 (2020), pp. 4805–4830. DOI: 10.1007/s10664-020-09868-x. URL: https://doi.org/10.1007/s10664-020-09868-x.

[29] Csaba Faragó, Péter Hegedüs, and Rudolf Ferenc. “Cumulative code churn: Impact on maintainability”. In: *15th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2015, Bremen, Germany, September 27-28, 2015*. Ed. by Michael W. Godfrey, David Lo, and Foutse Khomh. IEEE Computer Society, 2015, pp. 141–150. DOI: 10.1109/SCAM.2015.7335410. URL: https://doi.org/10.1109/SCAM.2015.7335410.

[30] Henning Femmer et al. “Rapid quality assurance with Requirements Smells”. In: *J. Syst. Softw.* 123 (2017), pp. 190–213. DOI: 10.1016/J.JSS.2016.02.047. URL: https://doi.org/10.1016/j.jss.2016.02.047.

[31] Neil M Ferguson et al. “Strategies for mitigating an influenza pandemic”. In: *Nature* 442.7101 (2006), pp. 448–452.

[32] Alessio Ferrari et al. “Detecting requirements defects with NLP patterns: an industrial experience in the railway domain”. In: *Empirical Software Engineering* 23.6 (2018), pp. 3684–3733.

[33] Anthony R Florita and Gregor P Henze. “Comparison of short-term weather forecasting models for model predictive control”. In: *HVAC&R Research* 15.5 (2009), pp. 835–853.

[34] Martin Fowler. *Refactoring - Improving the Design of Existing Code*. Addison Wesley object technology series. Addison-Wesley, 1999. ISBN: 978-0-201-48567-7. URL: http://martinfowler.com/books/refactoring.html.

[35] Wei Fu, Tim Menzies, and Xipeng Shen. "Tuning for software analytics: Is it really necessary?" In: *Inf. Softw. Technol.* 76 (2016), pp. 135–146. DOI: 10.1016/J.INFSOF.2016.04.017. URL: https://doi.org/10.1016/j.infsof.2016.04.017.

[36] Takafumi Fukushima et al. "An empirical study of just-in-time defect prediction using cross-project models". In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. 2014, pp. 172–181.

[37] Emanuele Gentili, Jonida Çarka, and Davide Falessi. "A Systematic Mapping Study on Impact Analysis". In: *Proceedings of the 19th International Conference on Software Technologies, ICSOFT 2024, Dijon, France, July 8-10, 2024*. Ed. by Hans-Georg Fill et al. SCITEPRESS, 2024, pp. 375–382. DOI: 10.5220/0012758200003753. URL: https://doi.org/10.5220/0012758200003753.

[38] Emanuele Gentili and Davide Falessi. "Characterizing Requirements Smells". In: *Product-Focused Software Process Improvement - 24th International Conference, PROFES 2023, Dornbirn, Austria, December 10-13, 2023, Proceedings, Part I*. Ed. by Regine Kadgien et al. Vol. 14483. Lecture Notes in Computer Science. Springer, 2023, pp. 387–398. DOI: 10.1007/978-3-031-49266-2_27. URL: https://doi.org/10.1007/978-3-031-49266-2%5C_27.

[39] A Shaji George. "When trust fails: Examining systemic risk in the digital economy from the 2024 crowdstrike outage". In: *Partners Universal Multidisciplinary Research Journal* 1.2 (2024), pp. 134–152.

[40] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of software engineering*. Prentice-Hall, Inc., 1991.

[41] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric statistical inference: revised and expanded*. CRC press, 2014.

[42] Luiz Alberto Ferreira Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. "Bug report severity level prediction in open source software: A survey and research opportunities". In: *Inf. Softw. Technol.* 115 (2019), pp. 58–78. DOI: 10.1016/J.INFSOF.2019.07.009. URL: https://doi.org/10.1016/j.infsof.2019.07.009.

[43] Alex Graves and Alex Graves. "Long short-term memory". In: *Supervised sequence labelling with recurrent neural networks* (2012), pp. 37–45.

[44] Aditya Grover, Ashish Kapoor, and Eric Horvitz. "A deep hybrid model for weather forecasting". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 379–386.

[45] Xiaodong Gu et al. “Do Bugs Propagate? An Empirical Analysis of Temporal Correlations Among Software Bugs”. In: *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark (Virtual Conference)*. Ed. by Anders Møller and Manu Sridharan. Vol. 194. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 11:1–11:21. DOI: 10.4230/LIPICS.ECOOP.2021.11. URL: https://doi.org/10.4230/LIPIcs.ECOOP.2021.11.

[46] Zhongxian Gu et al. “Has the bug really been fixed?” In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*. Ed. by Jeff Kramer et al. ACM, 2010, pp. 55–64. DOI: 10.1145/1806799.1806812. URL: https://doi.org/10.1145/1806799.1806812.

[47] Tracy Hall et al. “A Systematic Literature Review on Fault Prediction Performance in Software Engineering”. In: *IEEE Trans. Software Eng.* 38.6 (2012), pp. 1276–1304. DOI: 10.1109/TSE.2011.103. URL: https://doi.org/10.1109/TSE.2011.103.

[48] J.A. Hanley. “The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve”. In: *Radiology* 143 (May 1982), pp. 29–36. DOI: 10.1148/radiology.143.1.7063747.

[49] Zellig S Harris. *Distributional structure*. 1954.

[50] Ahmed E. Hassan. “Predicting faults using the complexity of code changes”. In: *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*. IEEE, 2009, pp. 78–88. DOI: 10.1109/ICSE.2009.5070510. URL: https://doi.org/10.1109/ICSE.2009.5070510.

[51] M. Elizabeth C. Hull, Ken Jackson, and Jeremy Dick, eds. *Requirements Engineering, Third Edition*. Springer, 2011. ISBN: 978-1-8499-6404-3. DOI: 10.1007/978-1-84996-405-0. URL: https://doi.org/10.1007/978-1-84996-405-0.

[52] INCOSE. *INCOSE systems engineering handbook*. John Wiley & Sons, 2023.

[53] “ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering”. In: *ISO/IEC/IEEE 29148:2018(E)* (2018), pp. 1–104. DOI: 10.1109/IEEESTD.2018.8559686.

[54] Gareth James et al. *An introduction to statistical learning: With applications in python*. Springer Nature, 2023.

[55] Yue Jiang, Bojan Cukic, and Tim Menzies. “Fault Prediction using Early Lifecycle Data”. In: *ISSRE 2007, The 18th IEEE International Symposium on Software Reliability, Trollhättan, Sweden, 5-9 November 2007*. IEEE Computer Society, 2007, pp. 237–246. DOI: 10.1109/ISSRE.2007.24. URL: https://doi.org/10.1109/ISSRE.2007.24.

[56] Mayumi Itakura Kamata and Tetsuo Tamai. "How Does Requirements Quality Relate to Project Success or Failure?" In: *15th IEEE International Requirements Engineering Conference, RE 2007, October 15-19th, 2007, New Delhi, India.* IEEE Computer Society, 2007, pp. 69–78. DOI: 10.1109/RE.2007.31. URL: https://doi.org/10.1109/RE.2007.31.

[57] Yasutaka Kamei and Emad Shihab. "Defect Prediction: Accomplishments and Future Challenges". In: *Leaders of Tomorrow Symposium: Future of Software Engineering, FOSE@SANER 2016, Osaka, Japan, March 14, 2016.* IEEE Computer Society, 2016, pp. 33–45. DOI: 10.1109/SANER.2016.56. URL: https://doi.org/10.1109/SANER.2016.56.

[58] Yasutaka Kamei et al. "A Large-Scale Empirical Study of Just-in-Time Quality Assurance". In: *IEEE Trans. Software Eng.* 39.6 (2013), pp. 757–773. DOI: 10.1109/TSE.2012.70. URL: https://doi.org/10.1109/TSE.2012.70.

[59] Matt J Keeling and Pejman Rohani. *Modeling infectious diseases in humans and animals.* Princeton university press, 2008.

[60] Hossein Keshavarz and Meiyappan Nagappan. "ApacheJIT: A Large Dataset for Just-In-Time Defect Prediction". In: *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022.* ACM, 2022, pp. 191–195. DOI: 10.1145/3524842.3527996. URL: https://doi.org/10.1145/3524842.3527996.

[61] Pavneet Singh Kochhar, Dinusha Wijedasa, and David Lo. *A Large Scale Study of Multiple Programming Languages and Code Quality.* 2016. DOI: 10.1109/SANER.2016.112. URL: https://doi.org/10.1109/SANER.2016.112.

[62] Lars Kotthoff et al. "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA". In: *J. Mach. Learn. Res.* 18 (2017), 25:1–25:5. URL: https://jmlr.org/papers/v18/16-261.html.

[63] Taek Lee et al. "Developer Micro Interaction Metrics for Software Defect Prediction". In: *IEEE Trans. Software Eng.* 42.11 (2016), pp. 1015–1035. DOI: 10.1109/TSE.2016.2550458. URL: https://doi.org/10.1109/TSE.2016.2550458.

[64] Steffen Lehnert. "A taxonomy for software change impact analysis". In: *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, EVOL/IWPSE 2011, Szeged, Hungary, September 5-6, 2011.* Ed. by Anthony Cleve and Romain Robbes. ACM, 2011, pp. 41–50. DOI: 10.1145/2024445.2024454. URL: https://doi.org/10.1145/2024445.2024454.

[65] Stefan Lessmann et al. "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings". In: *IEEE Trans. Software Eng.* 34.4 (2008), pp. 485–496. DOI: 10.1109/TSE.2008.35. URL: https://doi.org/10.1109/TSE.2008.35.

[66] VI Levenshtein. “Binary codes capable of correcting deletions, insertions, and reversals”. In: *Proceedings of the Soviet physics doklady* (1966).

[67] Patrick Li. *Jira Software Essentials: Plan, track, and release great applications with Jira Software*. Packt Publishing Ltd, 2018.

[68] Zhiqiang Li et al. “An empirical study of data sampling techniques for just-in-time software defect prediction”. In: *Autom. Softw. Eng.* 31.2 (2024), p. 56. DOI: 10.1007/S10515-024-00455-8. URL: https://doi.org/10.1007/s10515-024-00455-8.

[69] Andrew W Lo and A Craig MacKinlay. *A non-random walk down Wall Street*. Princeton University Press, 2011.

[70] Edward N Lorenz. “Deterministic nonperiodic flow”. In: *Journal of atmospheric sciences* 20.2 (1963), pp. 130–141.

[71] Shinsuke Matsumoto et al. “An analysis of developer metrics for fault prediction”. In: *Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE 2010, Timisoara, Romania, September 12-13, 2010*. Ed. by Tim Menzies and Günes Koru. ACM, 2010, p. 18. DOI: 10.1145/1868328.1868356. URL: https://doi.org/10.1145/1868328.1868356.

[72] Shane McIntosh and Yasutaka Kamei. “Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction”. In: *IEEE Trans. Software Eng.* 44.5 (2018), pp. 412–428. DOI: 10.1109/TSE.2017.2693980. URL: https://doi.org/10.1109/TSE.2017.2693980.

[73] Seetaram Ruthvik Mugu et al. “Lessons from the CrowdStrike Incident: Assessing Endpoint Security Vulnerabilities and Implications”. In: *2024 Cyber Awareness and Research Symposium (CARS)*. IEEE. 2024, pp. 1–10.

[74] Olugbenro Ogundipe and Tejiri Aweto. “The shaky foundation of global technology: A case study of the 2024 crowdstrike outage”. In: *International Journal of Multidisciplinary Research and Growth Evaluation* 5.5 (2024), pp. 106–108.

[75] David Orrell et al. “Model error in weather forecasting”. In: *Nonlinear processes in geophysics* 8.6 (2001), pp. 357–371.

[76] Rana Ozakinci and Ayça Tarhan. “Early software defect prediction: A systematic map and review”. In: *J. Syst. Softw.* 144 (2018), pp. 216–239. DOI: 10.1016/J.JSS.2018.06.025. URL: https://doi.org/10.1016/j.jss.2018.06.025.

[77] Harsh Patel, Bram Adams, and Ahmed E. Hassan. “Post deployment recycling of machine learning models”. In: *Empir. Softw. Eng.* 29.4 (2024), p. 100. DOI: 10.1007/S10664-024-10492-2. URL: https://doi.org/10.1007/s10664-024-10492-2.

[78] Dewayne E. Perry, Harvey P. Siy, and Lawrence G. Votta. "Parallel changes in large-scale software development: an observational case study". In: *ACM Trans. Softw. Eng. Methodol.* 10.3 (2001), pp. 308–337. DOI: 10.1145/383876.383878. URL: https://doi.org/10.1145/383876.383878.

[79] Martin Pinzger, Nachiappan Nagappan, and Brendan Murphy. "Can developer-module networks predict failures?" In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*. Ed. by Mary Jean Harrold and Gail C. Murphy. ACM, 2008, pp. 2–12. DOI: 10.1145/1453101.1453105. URL: https://doi.org/10.1145/1453101.1453105.

[80] pmd. *PMD quickstart ruleset*. https://github.com/pmd/pmd/blob/main/pmd-java/src/main/resources/rulesets/java/quickstart.xml (Feb. 2025).

[81] Aleksandra Revina, Krisztián Búza, and Vera G. Meister. "IT Ticket Classification: The Simpler, the Better". In: *IEEE Access* 8 (2020), pp. 193380–193395. DOI: 10.1109/ACCESS.2020.3032840. URL: https://doi.org/10.1109/ACCESS.2020.3032840.

[82] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5 (1988), pp. 513–523.

[83] Emad Shihab et al. "Predicting Re-opened Bugs: A Case Study on the Eclipse Project". In: *17th Working Conference on Reverse Engineering, WCRE 2010, 13-16 October 2010, Beverly, MA, USA*. Ed. by Giuliano Antoniol, Martin Pinzger, and Elliot J. Chikofsky. IEEE Computer Society, 2010, pp. 249–258. DOI: 10.1109/WCRE.2010.36. URL: https://doi.org/10.1109/WCRE.2010.36.

[84] Robert H Shumway et al. "ARIMA models". In: *Time series analysis and its applications: with R examples* (2017), pp. 75–163.

[85] Kelly Smith. "Managing electronic resource workflows using ticketing system software". In: *Serials Review* 42.1 (2016), pp. 59–64.

[86] Liyan Song and Leandro L. Minku. "A Procedure to Continuously Evaluate Predictive Performance of Just-In-Time Software Defect Prediction Models During Software Development". In: *IEEE Trans. Software Eng.* 49.2 (2023), pp. 646–666. DOI: 10.1109/TSE.2022.3158831. URL: https://doi.org/10.1109/TSE.2022.3158831.

[87] Weifeng Sun et al. "Method-Level Test-to-Code Traceability Link Construction by Semantic Correlation Learning". In: *IEEE Trans. Software Eng.* 50.10 (2024), pp. 2656–2676. DOI: 10.1109/TSE.2024.3449917. URL: https://doi.org/10.1109/TSE.2024.3449917.

[88] Souhaib Ben Taieb et al. "Long-term prediction of time series by combining direct and MIMO strategies". In: *International Joint Conference on Neural Networks, IJCNN 2009, Atlanta, Georgia, USA, 14-19 June 2009*. IEEE Computer Society, 2009, pp. 3054–3061. DOI: 10.1109/IJCNN.2009.5178802. URL: https://doi.org/10.1109/IJCNN.2009.5178802.

[89] Chakkrit Tantithamthavorn, Ahmed E. Hassan, and Kenichi Matsumoto. "The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models". In: *IEEE Trans. Software Eng.* 46.11 (2020), pp. 1200–1219. DOI: 10.1109/TSE.2018.2876537. URL: https://doi.org/10.1109/TSE.2018.2876537.

[90] Chakkrit Tantithamthavorn et al. "An empirical comparison of model validation techniques for defect prediction models". In: *IEEE Transactions on Software Engineering* 43.1 (2016), pp. 1–18.

[91] Chakkrit Tantithamthavorn et al. "The Impact of Automated Parameter Optimization on Defect Prediction Models". In: *IEEE Trans. Software Eng.* 45.7 (2019), pp. 683–711. DOI: 10.1109/TSE.2018.2794977. URL: https://doi.org/10.1109/TSE.2018.2794977.

[92] Burak Turhan and Tim Menzies and Ayse Basar Bener and Justin S. Di Stefano. "On the relative value of cross-company and within-company data for defect prediction". In: *Empirical Software Engineering* 14.5 (2009), pp. 540–578.

[93] Qasim Umer, Hui Liu, and Inam Illahi. "CNN-Based Automatic Prioritization of Bug Reports". In: *IEEE Trans. Reliab.* 69.4 (2020), pp. 1341–1354. DOI: 10.1109/TR.2019.2959624. URL: https://doi.org/10.1109/TR.2019.2959624.

[94] Andric Valdez et al. "Sentiment Analysis in Jira Software Repositories". In: *2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT)*. 2020, pp. 254–259. DOI: 10.1109/CONISOFT50191.2020.00043.

[95] Bailey Vandehei, Daniel Alencar da Costa, and Davide Falessi. "Leveraging the Defects Life Cycle to Label Affected Versions and Defective Classes". In: *ACM Trans. Softw. Eng. Methodol.* 30.2 (2021), 24:1–24:35.

[96] Chuanqi Wang et al. "Examining the effects of developer familiarity on bug fixing". In: *J. Syst. Softw.* 169 (2020), p. 110667. DOI: 10.1016/J.JSS.2020.110667. URL: https://doi.org/10.1016/j.jss.2020.110667.

[97] Karl E Wiegers and Joy Beatty. *Software requirements*. Pearson Education, 2013.

[98] Frank Wilcoxon. "Individual comparisons by ranking methods". In: *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 196–202.

[99] William M. Wilson, Linda H. Rosenberg, and Lawrence E. Hyatt. "Automated Analysis of Requirement Specifications". In: *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.* Ed. by W. Richards Adrion et al. ACM, 1997, pp. 161–171. DOI: 10.1145/253228.253258. URL: https://doi.org/10.1145/253228.253258.

[100] Emily Winter et al. "How do Developers Really Feel About Bug Fixing? Directions for Automatic Program Repair". In: *IEEE Trans. Software Eng.* 49.4 (2023), pp. 1823–1841. DOI: 10.1109/TSE.2022.3194188. URL: https://doi.org/10.1109/TSE.2022.3194188.

[101] Ian H Witten and Eibe Frank. "Data mining: practical machine learning tools and techniques with Java implementations". In: *Acm Sigmod Record* 31.1 (2002), pp. 76–77.

[102] Claes Wohlin. "Guidelines for snowballing in systematic literature studies and a replication in software engineering". In: *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014.* Ed. by Martin J. Shepperd, Tracy Hall, and Ingunn Myrtveit. ACM, 2014, 38:1–38:10. DOI: 10.1145/2601248.2601268. URL: https://doi.org/10.1145/2601248.2601268.

[103] Claes Wohlin et al. *Experimentation in Software Engineering, Second Edition.* Springer, 2024. ISBN: 978-3-662-69305-6. DOI: 10.1007/978-3-662-69306-3. URL: https://doi.org/10.1007/978-3-662-69306-3.

[104] Hongyu Zhang. "An investigation of the relationships between lines of code and defects". In: *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada.* IEEE Computer Society, 2009, pp. 274–283. DOI: 10.1109/ICSM.2009.5306304. URL: https://doi.org/10.1109/ICSM.2009.5306304.

[105] Lei Zhang and Bing Liu. "Sentiment Analysis and Opinion Mining". In: *Encyclopedia of Machine Learning and Data Mining.* Ed. by Claude Sammut and Geoffrey I. Webb. Springer, 2017, pp. 1152–1161. DOI: 10.1007/978-1-4899-7687-1_907. URL: https://doi.org/10.1007/978-1-4899-7687-1%5C_907.

[106] Tao Zhang et al. "Towards more accurate severity prediction and fixer recommendation of software bugs". In: *J. Syst. Softw.* 117 (2016), pp. 166–184. DOI: 10.1016/J.JSS.2016.02.034. URL: https://doi.org/10.1016/j.jss.2016.02.034.

[107] Yunhua Zhao, Kostadin Damevski, and Hui Chen. "A Systematic Survey of Just-in-Time Software Defect Prediction". In: *ACM Comput. Surv.* 55.10 (2023), 201:1–201:35. DOI: 10.1145/3567550. URL: https://doi.org/10.1145/3567550.

[108] Yu Zhou et al. "Combining text mining and data mining for bug report classification". In: *J. Softw. Evol. Process.* 28.3 (2016), pp. 150–176. DOI: `10.1002/SMR.1770`. URL: `https://doi.org/10.1002/smr.1770`.

List of Figures