Requirements Engineering (2022) 27:231–248
https://doi.org/10.1007/s00766-022-00372-w

ORIGINAL ARTICLE

An impact-driven approach to predict user stories instability

Yarden Levy[1] · Roni Stern[1,2] · Arnon Sturm[1] · Argaman Mordoch[1] · Yuval Bitan[1]

Received: 3 February 2021 / Accepted: 14 January 2022 / Published online: 18 March 2022

Abstract
A common way to describe requirements in Agile software development is through *user stories*, which are short descriptions of desired functionality. Nevertheless, there are no widely accepted quantitative metrics to evaluate user stories. We propose a novel metric to evaluate user stories called *instability*, which measures the number of changes made to a user story after it was assigned to a developer to be implemented in the near future. A user story with a high instability score suggests that it was not detailed and coherent enough to be implemented. The instability of a user story can be automatically extracted from industry-standard issue tracking systems such as Jira by performing retrospective analysis over user stories that were fully implemented. We propose a method for creating prediction models that can identify user stories that will have high instability even before they have been assigned to a developer. Our method works by applying a machine learning algorithm on implemented user stories, considering only features that are available before a user story is assigned to a developer. We evaluate our prediction models on several open-source projects and one commercial project and show that they outperform baseline prediction models.

Keywords User story · Requirements · Agile software development · Machine learning

1 Introduction

Agile is a widely used approach for software development. Most Agile software development methodologies [3–5] are characterized by short development cycles. Each cycle, often referred to as a *sprint*, focuses on implementing a small set of requirements.

A key challenge in agile development methodologies is the elicitation and communication of these requirements [42]. On the one hand, requirements aim to be detailed enough so that they are properly implemented. On the other hand, agile methodologies often stress that requirement specifications must be concise so that the overall development process is rapid.

One common practice to balance these contradictory desires is through *User Stories*. A User Story (US) is "a small piece of functionality of the final software perceived from the end-user point of view" [37]. The widespread adoption of user stories in software development projects [16, 24, 30, 42] introduces the challenge of how to assess their quality. Indeed, poorly written, low-quality, user stories may result in erroneous implementation, delays, and additional costs [42]. Prior studies followed a *text-based approach* to evaluate user stories, where a user story is evaluated by analyzing its textual description and checking if it satisfies a set of pre-defined desirable and undesirable properties. For example, Lucassen et al. [29] suggested 14 criteria to determine the quality of USs. These criteria include (1) the formation of a correct full sentence, (2) atomicity, i.e., to express exactly one feature, and (3) uniformity, i.e., that all user stories follow roughly the same template. Manual and automated tools were proposed for evaluating user stories by checking if the textual description of a given user story satisfies similar properties [25, 27]. However, it has not been established empirically that user stories with these properties are indeed more useful to the development team than user stories that do not [32].

✉ Arnon Sturm
sturm@bgu.ac.il

Yarden Levy
levyyard@post.bgu.ac.il

Roni Stern
sternron@post.bgu.ac.il

Argaman Mordoch
mordocha@post.bgu.ac.il

Yuval Bitan
ybitan@bgu.ac.il

[1] Ben Gurion University of the Negev, Beer-Sheva, Israel

[2] Palo Alto Research Center (PARC), Palo Alto, USA

In this paper, we propose a complementary approach in which a user story is assessed by how it impacts the development process. Indirect evidence about the impact of a user story on the development process can be obtained by analyzing data collected automatically from Jira,[1] which is an industry-standard *issue tracking system*. We propose several user-story quality metrics based on these data and discuss the pros and cons of each metric. Then, we focus on one such metric that we call *instability*. Instability refers to the number of changes made to a user story after it has been *assigned to a sprint*. Assigning a user story to a sprint means that the development team is committed to implementing it in the immediate future (usually 2–4 weeks). Thus, user stories at this stage are expected to be detailed and clear enough for implementation. Consequently, changes to a user story after it has been assigned to a sprint are considered inefficient [34] and may indicate that the user story has been poorly written. In general, changes to requirements have been identified as a major challenge in requirement engineering [35].

A major limitation of instability is that they are only known post-hoc, i.e., after the user story is closed. Ideally, we would like to assess the quality of a user story *before* it is added to a sprint to be implemented. To address this challenge, we developed a method for creating models that predict the instability of a user story before it is assigned to a sprint. This method involves applying machine learning techniques over training data automatically extracted from user stories that have already been implemented. A suite of features is proposed for this purpose, including features based on analyzing the textual description of the user story, features based on the history of the user story author, and process-based features.

We evaluate the performance of the proposed instability prediction models and the importance of the different features they use on several real-world projects, including open-source projects and one commercial project. We followed a within-project design, creating a prediction model for each project using historical data from that project.[2] The results indicate that the models predict user story instability more accurately than several baselines, whereas the impact of the different features varies between projects.

The contributions of this work are the following:

- Defining the notion of user story instability.
- Developing a method to generate models that can predict the instability of a user story before it is assigned to a sprint.
- Creating a dataset of user story issues from real-world software projects.
- Evaluating the accuracy of the generated prediction models on real-world software projects.

We made all the non-proprietary data in our dataset publicly available at https://doi.org/10.5281/zenodo.5579864 to allow reproducing our results. To this end, we also made our code publicly available at https://doi.org/10.5281/zenodo.5822986, which includes detailed instructions on how to reproduce our results.

The practical implications of this work are threefold. First, by measuring the instability of user stories over time, one can identify user story authors that tend to write unstable user stories and thus may need additional training or supervision. Second, when a person authors a user story, our instability prediction method provides continuous and predictive feedback regarding the quality of the user story that is created. In particular, providing authors with the predicted instability of their user stories can help reduce changes to user stories after they are assigned to a sprint. Third, deeper analysis of unstable user stories may lead to a set of new guidelines on how to write and manage such user stories to better support the development process. Future research can study the relation between user story instability and other software development metrics such as development pace and defects ratio.

The rest of the paper is organized as follows: Sect. 2 reviews related studies, analyzing their strengths and limitations. Section 3 describes several metrics for user stories, including user story instability. Section 4 presents our approach for predicting user story instability. Section 5 presents and analyzes our experimental results. Section 6 discusses threats to validity, and finally, Sect. 7 concludes and sets plans for future research.

2 Related work

Zhao et al. [44] mapped recent NLP studies related to requirements engineering. According to their mapping, our work can be attributed to the analysis phase, where a new solution is developed through a field experiment (utilizing open and commercial projects data). Most tools from that category refer to traditional linguistics analysis and rules for detecting various problems in textual requirements. In this work, we aim to detect unstable users stories using machine learning and only rely on a limited form of linguistics analysis to extract features. To the best of our knowledge, this is a novel approach. Nevertheless, our work is related to two existing lines of research: (1) evaluating user story quality, and (2) predicting software development metrics.

[1] http://www.atlassian.com/software/jira/.

[2] That is, the data for each project were split to train and test. The train part was used to create a prediction model that was then evaluated on the test part of the data for that project.

2.1 Evaluating user stories quality

According to Leffingwell [27], a US needs to capture the essential elements of a requirement: who used it for, what the desired functionality is, and why it is important. To evaluate USs, Leffingwell defined a set of criteria called INVEST: Independent, Negotiable, Valuable to users or customers, Estimable, Small, and Testable [27]. These criteria are very broad and serve as a basis to other quality models which extend and add measures to these six criteria.

Lai [25] defined three factors for evaluating US quality: Basic Quality (Discussable & Estimable), Management Quality (Controllable & Manageable), and Acceptance Quality (Confirmable). Each factor consists of three metrics, with different weights, and the final quality metric is the weighted sum of all three factors. They also suggested rules for correcting low-quality USs.

Lucassen et al. [29] presented the Quality User Story (QUS) framework, which is a collection of 14 criteria that determine the quality of USs. Using this framework, they developed a tool called AQUSA [31], which uses NLP techniques to verify some of the QUS criteria automatically. Each quality criteria is classified according to three concepts from linguistics: syntactic, semantic, and pragmatic. The QUS framework and the AQUSA tool have been combined into a package called the Grimm Method [32], which integrates QUS and AQUSA into the development process during US creation, planning, and implementation. They also evaluated the effect of using the Grimm Method on project management metrics including data extracted from JIRA, such as the number of defects created and development velocity. These metrics are similar to the impact-driven metrics discussed in this work. Their results, however, were inconclusive, showing no statistically significant impact on project management metrics when using their method.

Femmer et al. [18] defined a set of "requirements smells", which are undesirable requirement properties, and created a system called Smella to detect these requirement smells automatically.

All the above studies analyze and classify requirements by defining rules over the textual description of the requirement or by following expert classification. The limitation of this approach is that often there is no empirical evidence that establishes the correctness of the defined rules, and they are given as axioms. For example, many studies recommend that user stories should follow the template of "As a <type of user>, I want <goal or objective>, [so that <some reason>." However, to the best of our knowledge, there is no empirical evidence that indicates USs that follow this standard template are more likely to be understood by the developers.

2.2 Predicting software development metrics

Predicting various types of software development metrics is a rapidly growing research field. For example, Hayes et al. used a decision tree to predict the testability of the requirements based on several readability measures [22]. They demonstrated the ability to predict requirement testability, but the validity of their results is limited due to the limited dataset they used.

Dragan et al. used requirements quality measures to predict the operational performance of the developed system [17]. Requirements quality was determined based on a linguistic analysis aimed towards measuring whether the requirements are incomplete, untestable, of weak phrase, vague, or compound. They examined four classifiers: logistic regression, K-Nearest Neighbor (KNN), Naive Bayes, and Support Vector Machine (SVN). They found out that SVN and logistic regression results in a random classifier and KNN achieved the best results of accuracy (86%).

Choetkiertikul et al. [12] developed a method to predict delays experienced when implementing USs. In their prediction model, they used text-based topic models, number of comments, number of effect versions, and additional process-based information from Jira. It also compares different prediction times of the development process and uses several machine learning classifiers. This work is reminiscent of our instability prediction task. However, delays in the implementation of a USI may result from a variety of reasons that are not related to US quality, such as an incorrect effort estimation or the complexity of the requested functionality.

Effort estimation in software projects is a widely studied prediction problem in which the task is to predict the development efforts for implementing a given requirement. Non-automated methods for effort estimation such as planning poker have been proposed and used in practice [21, 33]. A variety of automated methods have been proposed for US effort estimation [45], including methods based on Deep Learning [2, 13, 36] and Dynamic Bayesian Networks [23]. The data used by these methods are extracted from the US text.

The effort of implementing a user story is often measured in *story points* as opposed to actual time. A story point is a *relative* unit of measure that is used by the development team "to provide relative estimates of effort for completing requirements" [14]. Choetkiertikul et al. [13] proposed a data-driven method to estimate the number of story points assigned to a US, using a deep learning method that contains a Long Short-Term Memory (LSTM) layer and a recurrent highway net. Shi et al. proposed a model to predict requirement changes based on historical changes [40]. The model consists of the topics, the number of changes, the number of versions, and the sequence of changes. They found out that following a regression model the various factors indeed

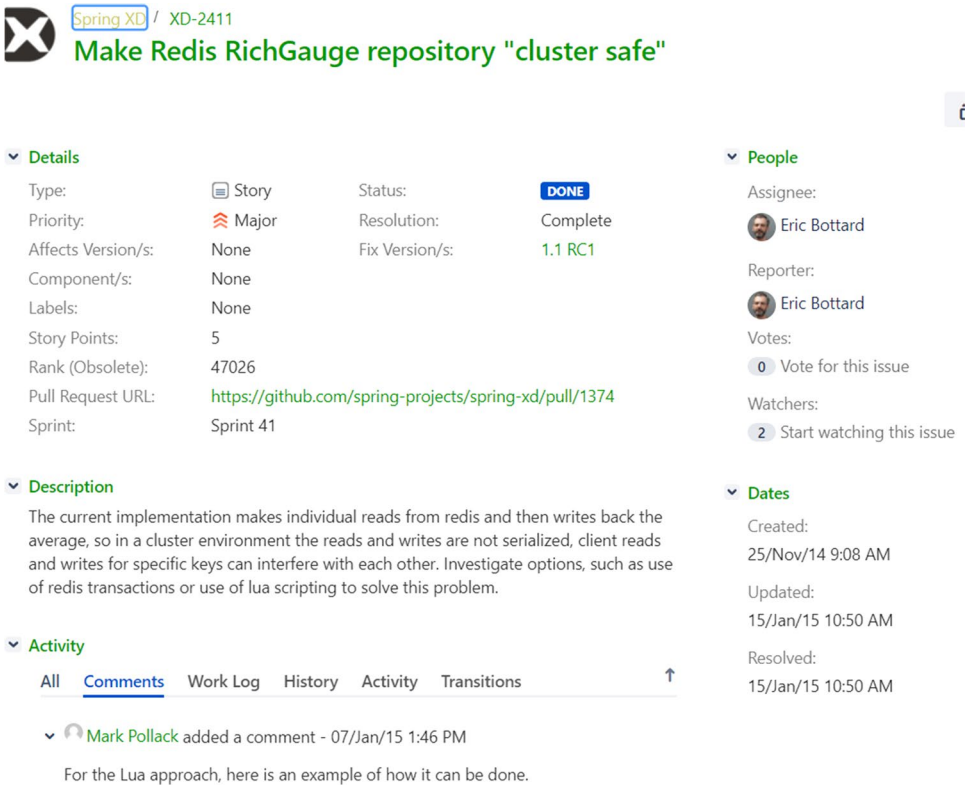

Fig. 1 A screenshot of USI XD-2411, as shown in the Jira web interface (publicly available at https://jira.spring.io/browse/XD-2411)

support the prediction. Nevertheless, the F-measure of the various models is quite limited (31%). In general, effort estimation is fundamentally different from US quality assessment, since a US that requires significant effort to implement is not necessarily poorly written.

3 Impact-driven US metrics

The main objective of writing a US is to concisely communicate a requirement. Poorly communicating a requirement is expected to make its implementation more difficult and have other negative effects on the development process. Therefore, we propose to evaluate USs in retrospect by analyzing US properties over the course of their development and implementation. We refer to metrics that evaluate user stories in this way as *impact-driven* US metrics since they aim to measure the impact of a low-quality user story on the development process. The impact-driven metrics we propose are described in the context of the Jira issue tracking tool and the popular *Scrum* software development methodology [38].

3.1 Using user stories in Jira and scrum

Software development in Scrum progresses through *sprints*, where every sprint is a "time-box of one month or less during which a "Done", usable, and potentially releasable product increment is created" [39] in Scrum is a set of *backlog items*, where every backlog item in Scrum represents a development task, e.g., implementing a user story or fixing a bug. The *product backlog* is a list of all backlog items that are of interest. Throughout the development process, backlog items are updated, added, and deleted from the *product backlog*. At the beginning of each sprint, a set of issues are added to the *sprint backlog*, with the intention of fully implementing them during the sprint. The duration of a sprint is fixed throughout the project, and a new sprint starts immediately after one finishes.

Scrum is supported by many tools, one of which is the Jira issue tracking system. Every backlog item in Scrum corresponds to a *Jira issue*. Jira allows associating types to issues, e.g., "Bug" and "User Story". This research deals with issues of the latter types, referred to as User Story Issues (USIs), to focus on user stories that describe new features or requirements that need to be developed as opposed to bug reports.

Note that this labeling of USIs is not perfect, and not every USI marked as a "User Story" is indeed a user story.

Figure 1 shows a screenshot from the Jira web interface, showing some of the data Jira stores about USI XD-2411 from the open-source Spring (XD) project. These data include the type of USI (in this case it is "Story"), a priority score (in this case, "Major"), the identity of the person that wrote this USI (the "Reporter" field), comments made by the

development team about this USI, and a textual description (the "Description" field).

3.2 Impact-driven metrics

Jira manages and stores an abundance of information about every USI, including when it was created, when it was added to a sprint, who wrote it, and its current status (e.g., implementation has not started, in progress, and completed). In addition, Jira tracks all the changes performed on every USI, marking when they occurred and what was changed. Based on these data, we propose and discuss several potential impact-driven USI metrics.

3.2.1 Time to completion

According to Scrum best practices, a USI should be a short requirement that can be completed during a single sprint. Thus, if implementing a USI requires more than one sprint, this could indicate that the requirement was too long, included too much functionality, or was badly written.

To quantify this observation, we consider the following impact-driven metric. Let x be a USI we want to evaluate, and let $sprints(x)$ denote the set of sprints that x was assigned to. That is, $sprints(x)$ is the set of sprints in which x was added to their sprint backlog, until the status of x has changed to "Done".

The first impact-driven metric we propose is $|sprints(x)|$, which estimates *the number of sprints required to implement a USI*. This metric can be extracted automatically from Jira.

The problem with this metric is that there are other, common, reasons why multiple sprints are needed to implement a USI. For example, a delay in the implementation of a higher priority USI assigned to the same sprint may cause the implementation of the related USI to also be delayed and "drag" to the subsequent sprint. While this is not desirable, it does not indicate that the specific USI has been poorly written.

3.2.2 Changes during a sprint

Like all Agile methodologies, Scrum is designed to "embrace change" [19]. However, Scrum strictly discourages changes to a USI after it is assigned to a sprint and a developer to be implemented. That is, while a USI is in the product backlog, the team should discuss it and change its description and other properties. However, a USI should enter a sprint backlog only after the team discussed it and agreed that it is sufficiently defined, and ready to be implemented [8]. Based on this, we propose two impact-driven metrics that measure changes made to a USI after it has been assigned to a sprint.

Effort estimation changes. This metric considers changes in effort estimation, namely changes made to the number of story points assigned to a USI. Such a change during a sprint may indicate the incorrect interpretation of the US, suggesting that the USI is poorly written.

The main limitation of this metric is similar to that of the number of sprints metric: there are other, common, reasons for why the number of story points may change during a sprint. Accurate effort estimation in software is notoriously difficult, even for a requirement that is well-written. For example, a change in story points may be due to incorrect estimation of the technical difficulty of implementing a task as opposed to a lacking description of the requirement. Also, the developer that made the effort estimation for USI x may be different from the one that wrote the USI.

Textual description changes. A different metric we propose is based on measuring changes to the textual description of the USI done after it has been assigned to a sprint.

Change in the textual description of a USI implies that it was not well defined, misunderstood, includes ambiguities, or lacks details. This does not mean that changes do not occur during implementation. However, since USs are rarely used for documentation purposes, the fact that a developer took the time to update a US during a sprint suggests that this change is required now to implement it. We call this metric *USI Instability*.

Definition 1 (*Instability*) The instability of USI x with respect to sprint y, denoted $instability(x, y)$, is the number of words in the textual description of x that have changed after x was assigned to y. A USI x is k-unstable if the sum of instabilities with respect to all sprints it was part of is at least k. That is, x is k-unstable iff

$$\sum_{y \in sprints(x)} instability(x, y) \geq k \tag{1}$$

A "change" here means that the word has either been added, deleted, replaced, or modified. We observed that it is often the case that the textual description of a USI is modified immediately after it has been added to a sprint. Such changes are most likely the result of discussions made just before adding the USI to the sprint. Thus, when computing the instability of a USI, we ignored changes made in the first hour after it has been added to the sprint.

Ideally, instability would be quantified by performing a semantic analysis of the changes done to the USI. However, such an analysis is very challenging since the USIs we consider do not all conform to a single standard such as the Connextra format for user stories. Thus, we leave to future research the development of an instability metric that applies such a semantic analysis.

3.3 USI instability in the wild

Since instability is a novel metric for USIs, we examine below its values in USIs from real-world projects.

3.3.1 Data collection

We collected USIs from the following open-source projects.

- *Alfresco (REPO)*[3] An enterprise-class, cloud-native content services platform.
- *JBossDeveloper (DEVELOPER)*[4] A developer environment that integrates with the JBoss application server.
- *LSST (DM)*[5] Software services for a vast astronomical dataset.
- *Spring (XD)*[6] An application framework and inversion of control container for the Java platform.

These projects were chosen because they use Jira and Scrum and have their Jira repository publicly available. In addition, we collected USIs from a software project in a large multinational corporation that specializes in software and services. We refer to this project as Commercial. Thus, our dataset comprised USIs from four open-source projects and one commercial project.

From each project, we only collected USIs have been completely implemented (i.e., marked as *closed*) and have been included in a sprint. These two conditions are needed to be able to compute instability. Table 1 presents descriptive statistics about the data we collected from the open-source projects. Specifically, Table 1 lists when each project was created (the "Created" column), the number of Jira issues in these projects—including bug issues ("Issues"), the number of sprints in which these issues were in ("Sprint"), and the average number of words in each issue ("Avg. Words"), ignoring stop words.

3.3.2 Changes in unstable USIs

Figures 2, 3, 4, and 5 show examples of unstable USIs from the REPO, DM, DEVELOPER, and XD projects. The left and right columns show the USI description as it was added to a sprint and when its implementation was complete, respectively. We highlighted in red the text that was deleted and in blue the text that was added. The instability of these USIs is 17, 58, 113, and 48 for Figs. 2, 3, 4, and 5, respectively.

[3] https://issues.alfresco.com/jira.

[4] https://issues.jboss.org.

[5] https://jira.lsstcorp.org.

[6] https://jira.spring.io.

Table 1 General statistics from the open-source projects in our dataset

Project	Created	Issues	Sprints	Avg. words
Alfresco (REPO)	2015	2457	138	64
JBossDeveloper (DEVELOPER)	2015	5379	129	55
LSST (DM)	2014	20,106	276	48
Spring (XD)	2013	3710	66	40

By examining these unstable USIs and others, we found that the changes made in these USIs can be classified into these categories: (1) refinement, i.e., adding more details; (2) adding missing information, e.g., acceptance tests; (3) adding rationale to the USI; and (4) rephrasing the USI. The changes done to most of the unstable USIs are refinements. In the examples above, the changes in the second and third USIs (Figs. 3, 4) are refinements. The changes in Fig. 3 also add rationale to the USI. The change in the fourth USI (Fig. 5) also includes rephrasing the USI description, so that it fits the Connextra user story template.

A deeper study of the changes that occur to a USI over time may contribute to characterizing the changes and cluster them into types in a more comprehensive manner. This is beyond the scope of this paper.

3.3.3 Results

Table 2 shows the number (and percentage) of 5-unstable, 10-unstable, 15-unstable, and 20-unstable USIs from all the projects in our dataset. The "USI" column shows the number of USI in each project.

The results show that most USIs are not 5-unstable. For example, from the 1,229 USIs collected from the DEVELOPER project, only 394 were 5-unstable. Note that if an USI is 10-unstable, then it is also 5-unstable. Thus, we can conclude that the instability of most USIs is less than 5 w.r.t to all their corresponding sprints.

This supports our rationale that making changes to a USI during a sprint is not the common practice and is not desirable.

A common conjecture in the literature is that correctly written user stories should be written in the well-known Connextra form: "As a (role) I want (something) so that (benefit)" [30]. As a side note, we checked if there is a correlation between the k-instability of a USI and the usage of this template. The column "All" in Table 3 shows the percentage of USIs that follow the Connextra template. The columns $k = 5$ and $k = 20$ under "with template" show the percentage of USIs that are k-unstable and follow the

Before	After
Summary: EOL of spring-social Description: EOL of spring-social-* end This story is to remove our dependencies on spring-social-*. Social features were removed in a previous release, but there may still be so code that still uses these libraries. We currently have the following libraries: * spring-social-core * spring-social-facebook * spring-social-facebook-web * spring-social-linkedin * spring-social-twitter * spring-social-web *Questions:* * TODO *Acceptance Criteria:* * Build green after removing libraries * TODO	Summary: EOL of spring-social Description: This story is to remove our dependencies on spring-social-*. Social features were removed in a previous release, but there may still be so code that still uses these libraries. We currently have the following libraries: * spring-social-core * spring-social-facebook * spring-social-facebook-web * spring-social-linkedin * spring-social-twitter * spring-social-web *Acceptance Criteria:* * Build green after removing libraries * Remove the associated tests * Update *l10n.properties* and notify L10n team

Fig. 2 USI REPO-1296 from the REPO project

Before	After
Summary: Make SpanSet operator templates more generic Description: make the template parameters more flexible such that ndarrays can be interpreted as array like or vector like instead of just 2 or 1 d arrays.	Summary: Make SpanSet operator templates more generic Description: Expand the flatten and unflatten methods of SpanSets such that they can operate on multi-dimensional ndarrays. This work involves making the template parameters for these functions, and the getter classes more generic.

Fig. 3 USI DM-9105 from the DM project

Before	After
Summary: Use Drupal for nightly builds Description: * URLs need to be fixed to not use `docker` * Acceptance tests need to be run against the nightly	Summary: Create a Jenkins job to build the Drupal Pilot env Description: This is the job that rebuilds the Drupal pilot environment. Initially set this to build weekly until the data is persisted (it will be annoying if people keep losing changes). Todos: * URLs need to be fixed to not use `docker` * Acceptance tests need to be run against the nightly Definition of Done * There is a URL that anyone on the VPN can use to visit the site * Acceptance tests are triggered after the build completes * All 'critical' acceptance tests pass (to be decided, once the list of failures is known) * All non-critical issues are JIRA'd

Fig. 4 USI DEVELOPER-1296 from the DEVELOPER project

Connextra template. Similarly, the columns $k = 5$ and $k = 20$ under "without template" show the percentage of USIs that are k-unstable and do not follow the Connextra template.

The results show several interesting trends. First, the vast majority of USIs does not follow the Connextra template. For example, in the DM project, only 0.1% of the USIs followed the template. Second, we do not observe any correlation between the USIs that follow the template and USIs that are 5-unstable or 20-unstable. For example, consider the DEVELOPER and REPO projects and $k = 5$. In DEVELOPER, 24% of the USIs that followed the template were 5-unstable, and 32% of the USIs that did not follow the template were 5-unstable. By contrast, in REPO, 34% of the

Before	After
Summary: Create a SFTP source Description: Use the SFTP adapter in SI as the basis. Need to consider the infrastructure for testing.',	Summary: Add SFTP source Description: As a user, I'd like to have the option to use the _SFTP_ source module so that I can access, transfer, and mange files over any reliable data streams. Reference: [Spring Integration SFTP Adapter|http://docs.spring.io/spring-tegration/reference/html/sftp.html] Need to consider the infrastructure for testing.

Fig. 5 USI XD-2044 from the XD project

Table 2 List of the number of the four types of *unstable* USI-changes in more than 5, 10, 15, and 20 words, in all the different projects (in brackets the percentage of these from all the USIs)

Project	USI	5-unstable	10-unstable	15-unstable	20-unstable
Commercial	1853	377 (20%)	301 (16%)	256 (14%)	235 (13%)
DEVELOPER	1229	394 (32%)	342 (28%)	319 (26%)	290 (24%)
REPO	900	195 (22%)	195 (22%)	172 (19%)	140 (16%)
XD	1741	173 (10%)	125 (7%)	99 (6%)	80 (5%)
DM	4962	626 (13%)	523 (11%)	467 (9%)	415 (8%)

Table 3 Percentage of USIs that follow ("with template" column) or do not follow ("without template" column) the recommended US template of "As a (role) I want (something) so that (benefit)"

Project	With template			Without template	
	All	$k = 5$ (%)	$k = 20$ (%)	$k = 5$ (%)	$k = 20$ (%)
Commercial	4.4	30.0	15.6	20.0	19.6
DEVELOPER	3.0	24.0	13.5	32.0	23.9
REPO	15.2	34.0	24.8	24.0	13.8
XD	17.0	20.0	11.4	8.0	3.1
DM	0.1	0.0	0.0	13.0	8.3

"All" column shows the percentage out of all USIs in our dataset. The values in the $k = 5$ and $k = 20$ columns show the percentage of USIs that are k-unstable out of either follow or do not follow the recommended US template

USIs that followed the template were 5-unstable, and 24% of the USIs that did not follow the template were 5-unstable. In conclusion, it seems that the well-known Connextra format is not common in the real-world USIs in our dataset, and it does not have a direct effect on the USI instability. A possible explanation for these two trends is that while some of the USIs in our dataset are labeled as "user stories", others are labeled as "change requests" and similar labels.[7]

4 Predicting user stories instability

To compute the instability of a USI, it must have already been assigned to a sprint (Definition 1). Similarly, to check if a USI x is k-unstable the status of x must have already been changed to "Done". Thus, instability can be used for retrospective evaluation of USIs but it cannot be directly used to assess USIs *before* they are passed to be implemented.

In this section, we introduce an approach based on Machine Learning (ML) for predicting whether a USI will be k-unstable based only on data available before it is being explicitly implemented.

4.1 Instability prediction as a supervised learning task

Supervised Learning is a branch of ML that includes algorithms and statistical models designed to predict future behavior based on past behavior. Specifically, supervised learning algorithms require information about past behavior in the form of labeled data, that is, a training set that comprises (instance, label) pairs, where the label is the expected prediction.

In our case, we apply supervised learning to solve the instability prediction task. An instance is a USI, and the label is whether that USI will be k-unstable, for some value of k. As a training set, we use the set of USIs that were already done in the project, since computing their label is straightforward (Def. 1).

Given such a training set, we adopt a standard supervised learning methodology that consists of extracting features

[7] USIs labeled as "defects" or "bug reports" are not included.

from each instance, choosing a supervised learning algorithm, and setting its hyper-parameters.

A major challenge in the application of supervised learning, in general, is how to identify features of an instance that are related to its expected label. Smart feature extraction is particularly important in our case since the training set size in our datasets is relatively small (see Table 1), and thus deep learning techniques that work directly on the raw data will not be effective.

4.2 Feature extraction for instability prediction

We propose four groups of features for the instability prediction task: simple text-processing features, advanced text-processing features, process features, and personalized features. The first two groups of features are based on analyzing the textual description of the USI at the time when the USI was assigned to a sprint.[8] As a pre-processing, we removed all HTML tags, URLs, and code snippets from the extracted textual description.

4.2.1 Simple text processing features

The features in this group are derived from the text without applying sophisticated Natural Language Processing (NLP) techniques.

- *Text length* The number of characters in the USI text. The rationale for this feature is that short text may suggest that details are missing, and very long text may suffer from over details, limited cohesion, and cause confusion.
- *Contains a URL* One if the USI text contains a URL, and zero otherwise.
- *Contains "missing details" words* One if the USI text contains the words "TODO/TBD/Please", and zero otherwise. These words imply something missing or unclear in the text and further editing is suggested.

4.2.2 Advanced text processing features

The features in this group are based on applying modern Natural Language Processing (NLP) techniques. Specifically, we apply topic modeling [41] and document embedding [26] to create quantitative representations of the USI text content and use these representations as features. Topic modeling and document embedding require training. In both cases, we trained a separate model for each project in our dataset, training on our training set and tuning the model's parameters (e.g., number of topics in the topic model) on a validation test, as opposed to the test set.

- *Topic modeling* Topic modeling is a standard technique in modern NLP. A topic is defined by a distribution over the corpus words, where words with a higher probability characterize the topic. For example, text that relates to medicine includes the words Doctor, illness, and disease in the top probability words.

  Given a topic model, we added a feature for each topic, and the value of the feature is how much the USI text is associated with the topic according to the topic model. The number of topics per project is a hyper-parameter that we tuned over a separate validation set. For the Commercial, DEVELOPER, and DM projects, the number of topics used is four, and for the XD and REPO projects, the number of topics used is three.
- *Document embedding* Distributed Memory Model of Paragraph Vectors (PV-DM) [26] is a generalization of word embedding designed to characterize the content of a paragraph or a document as a fixed-sized vector of real numbers. In our case, we trained a PV-DM model and used it to represent all the description text of a single USI as a vector of fixed size. This vector is used as a feature.

The rationale for these features is that the quality and instability of a USI may be related to the requirement it describes, e.g., important modules in the developed software may get more attention, and hence are expected to be of high quality and fewer changes will be introduced after entering a sprint.

4.2.3 Process metrics

The features in this group are based on analyzing the process a USI goes through until it is added to a sprint, including the changes made to its text, the discourse with the developer about it as recorded in the Jira comments, and the priority assigned to that USI. These data are available since Jira documents all the changes related to the USI from its creation. In more detail, the features in this group are as follows:

- *Number of changes in the text before entering a sprint* The value of this feature is calculated similar to the instability calculation, except that here we count only changes that happened before the USI is assigned to a sprint.
- *Number of comments before entering a sprint* Jira supports adding comments to a USI. This feature counts only comments that were added before the USI is assigned to a sprint.
- *USI priority* In Jira, every USI has a *priority* field. This feature is the value of this field at the time the USI is assigned to a sprint.

The rationale for this group of features is that many changes or comments may indicate a problem with the USI and suggest additional changes after it enters a sprint. In addition,

[8] The textual description of a USI was collected from the "summary", "description", and "acceptance criteria" fields in JIRA.

Fig. 6 Separation to training, validation, and test sets along the time of assigning USIs to a sprint

a high-priority USI may suggest that it has been written in haste and thus may require additional changes even after entering a sprint.

4.2.4 Personalized metrics

The features in this group are based on data about the author of the USI, including data available in Jira about the previous USIs the person has authored.

- *Author experience* The value of this feature is the number of USIs that the author of this USI has written before the current USI.
- *Author past instability* The value of this feature is the ratio of unstable USIs out of all the USIs the author wrote before the current USI.

The rationale for the first feature is that we expect experienced USI authors to produce higher-quality USIs. The rationale for the second feature is that we expect a USI author that usually produces unstable USI to continue to do so in the future, and vice versa. "Appendix 1.1" lists all the features we used in our experiments.

4.3 Implementation details

We implemented the learning approach described above, using our dataset to train models for predicting if a USI is 5-unstable and to train models for predicting if a USI is 20-unstable. To account for the different characteristics of each project, we created separate prediction models for each project. We describe the details of our implementation below.

4.3.1 Data collection

For every USI in our dataset, we computed its instability value and appropriate label (5-unstable and 20-unstable). Then, we split these labeled USIs from each project into three sets: a training set, a validation set, and a test set. The training and validation sets were used to generate the instability prediction models, where the training set was used to train the models and the validation set was used for features selection and hyper-parameters tuning. The test set is used later in Sect. 5 to evaluate the performance of the resulting prediction models.

Table 4 List the number and ratio of 5-unstable and 20-unstable USIs in our training and validation sets and in our test set, for the different projects in our dataset

Project	# of USI	5-unstable	20-unstable
Training and validation			
Commercial	1483	335 (23%)	217 (15%)
DEVELOPER	983	334 (34%)	247 (25%)
REPO	720	180 (25%)	116 (16%)
XD	1393	122 (9%)	49 (4%)
DM	3970	509 (13%)	332 (8%)
Test			
Commercial	370	40 (11%)	17 (5%)
DEVELOPER	246	60 (24%)	43 (17%)
REPO	180	48 (27%)	24 (13%)
XD	348	51 (15%)	31 (9%)
DM	992	117 (12%)	83 (8%)

To simulate the use of our predictor in an actual development process, we did not perform a *k*-fold cross-validation to select these three sets.[9] Instead, we sort the data by the time the USI had been assigned to a sprint and use the first set of USIs for the training set, the subsequent set of USIs for the validation set, and the last set of USIs for the test set.[10] For each project, we found the points in time for this separation such that the train, validation, and test sets consist of approximately 60%, 20%, and 20% of the data, respectively. Figure 6 illustrates this dataset separation along the sprint assignment time.

[9] Performing a *k*-fold cross-validation would require for some folds to use data for evaluation that was collected before the data for training was used. This is problematic, especially since the features in the "Personalized Metrics" family rely on analyzing the instability of USIs done in the past.

[10] Note that we have also repeated our experiments using a *k*-fold cross-validation, and the results obtained were similar to those reported here.

Table 5 χ^2 results for the categorial and ordinal features

Feature	Commercial	DEVELOPER	REPO	XD	DM
Contains code	–	–	–	–	✓
Contains URL	✓	–	–	–	–
Follows Connextra template	–	✓	✓	✓	–
Contains "TBD"	✓	–	–	✓	–
Contains "please"	–	–	–	–	✓
USI priority	✓	–	–	–	–
Has acceptance criteria headline	–	✓	✓	✓	–
Is "Description" field empty	✓	✓	✓	–	✓
Is "Acceptance" field empty	✓	–	–	–	–

4.4 Descriptive statistics

Table 4 presents the number and percentage of USIs that were 5-unstable and 20-unstable in our training and validation sets and in our test set, for each project. As can be seen, the class distribution in our training and test sets is not the same. The most extreme case is in our Commercial project, where the percentage of 5-unstable USIs is 23% in the training set and only 11% in the test set. This type of difference is not a consistent bias: in the "XD" project, the situation is reversed, where the percentage of 5-unstable USIs in the training set is larger than in the test set (15% vs. 9%). These differences in class distribution between the training set and the test set affect the prediction task and make it more challenging.

4.4.1 Learning algorithms

We experimented with three state-of-the-art supervised learning algorithms that are commonly used for classification problems similar to ours.

- *Random Forest (RF)* is an ensemble of independent decision trees [7].
- *Neural Network (NN)* is a nonlinear model that consists of an input layer, hidden layers, and output layer [6]. In this study, we used a multi-layer feed forward NN.
- *eXtreme Gradient Boosting (XGBoost)* is an implementation of the Gradient Boosted Decision Trees algorithm [10].

Each algorithm has its characteristics and advantages, which can fit our problem. NN can learn nonlinear and complex patterns in the data and can handle large datasets. RF and XGBoost are state-of-the-art tree-based algorithms that often outperform NN, especially for relatively small datasets such as ours.

Each algorithm has several hyper-parameters that affect its performance. For RF, these hyper-parameters include the number of trees, max features, and max depth of each tree to avoid over-fitting. For NN, these hyper-parameters include the number of hidden layer sizes, the number of neurons in each layer, the activation functions, the solver, and the learning rate. For XGboost, these hyper-parameters include the number of trees, max depth of each tree, and the scale of the positive weight (a parameter designed to deal with imbalanced datasets). To select the hyper-parameters, we performed standard parameters optimization, optimizing for the area under the precision-recall curve over the validation set.

We used the validation set to tune every algorithm separately for each project and each label ($k = 5$ or $k = 20$).

Since our dataset is imbalanced, we also experimented with under-sampling techniques, but they did not yield significant benefits. For comparison purposes, we also implemented two baseline prediction algorithms: random prediction (denoted Rnd) and a fixed prediction model that predicts that all USIs are not k-unstable (denoted Maj). The latter baseline represents classifying according to the majority label.

4.4.2 Feature selection

We performed the following feature selection process for each project over its validation set. First, we computed a χ^2 test to all the categorical and ordinal features, and only kept features that were shown to have a relation with the label. Table 5 presents the results of this step. The rows represent the features and the columns represent the projects. A "✓" sign means that we reject the null hypothesis that no relationship exists between the chosen feature and the instability label, with a significance of 0.05 (p value smaller than 0.05). This was the case, for example, for the "Contains code" feature in the DM project.

Table 6 The feature groups with the best results on the validation set for each project

Project	Simple text	Advanced text	Process	Personalized
Commercial	✓	✓		✓
DEVELOPER	✓		✓	✓
REPO			✓	✓
XD			✓	✓
DM	✓	✓	✓	✓

Then, we chose the subset of feature groups that maximizes the area under the ROC curve (AUC ROC).[11] The chosen feature groups for each project are presented in Table 6. Again, the rows represent the projects, the columns represent the four feature groups, and a "✓" sign inside the table represents that a group of features has been chosen. For example, all four groups were used in the prediction model for the DM project.

The results in Table 6 show that for all projects, the best results were obtained when using a combination of features from multiple feature groups. This highlights that although some individual features have a non-negligible relationship with our instability label (as shown in Table 5), no single feature should be used alone. One may explore adding individual single-feature decision rules as a pre-processing stage before using our prediction model, but supplying all relevant features to the learning algorithms should allow finer-grained use of the selected features.

5 Experimental evaluation

We conducted an experimental evaluation of the prediction models described above. This section describes this experimental evaluation design and its results.

5.1 Research questions and hypotheses

The main research questions we aim to check in this study are the following:

- *RQ1* Can we predict the USI's instability?
- *RQ2* Which features affect the USI's instability and to what extent?

Our main hypothesis (denoted H1) is that our instability prediction models will be more accurate than the baseline prediction models (Rnd and Maj), as they are taking advantage of many facets of the USI. An additional hypothesis (denoted H2) is that features from all of the four feature groups we proposed will be important, as they embody different potential reasons for instability. Table 6 already provides some support for H2, but a deeper analysis is given in Sect. 5.4.

[11] See the definition of AUC ROC in Sect. 5.2.

5.2 Evaluation metrics

Since the prediction models we want to evaluate are binary classifiers, we evaluate them by considering the number of True Positives (TP), False Negatives (FN), False Positives (FP), and True Negatives (TN) over the test set. In our context, TP refers to USIs that are k-unstable and predicted as such by the model; TN refers to USIs that are not k-unstable and predicted as such by the model, FP refers to USIs that are not k-unstable but are predicted as such by the model, and FN refers to USIs that are k-unstable but and are not predicted as such by the model.

We use these four values to compute three standard classifier metrics: Accuracy (denoted Acc), Area Under Curve of the Receiving Operating Characteristics (denoted AUC ROC), and Area Under Curve of the Precision-Recall Curve (denoted AUC PRC). For completeness, we provide here a brief description of these well-known metrics.

The area under the receiver operating characteristic (AUC ROC) curve is a value that represents the trade-off between the true positive rate (TPR) and false-positive rate (FPR) at various thresholds, where TPR is defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

and FPR is defined as:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{3}$$

A larger area under the ROC curve indicates better prediction. When the area under the curve is 1, it means a perfect model. When the area under the curve is 0.5, it means the model cannot distinguish between the instability cases.

The area under the precision-recall curves (AUC PRC) is a value that represents the trade-off between Precision and Recall using different probability thresholds, where Recall is the TPR and Precision is defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

Here again, a larger area under the precision-recall curve indicates better prediction. This metric is particularly useful in our case since it takes into account the imbalance of the two instability labels.

A poor classifier will be a horizontal line on the plot with a precision that is proportional to the number of positive

Table 7 Accuracy, AUC ROC, and AUC PRC results for the 5-instability prediction task

Project	AUC PRC					AUC ROC					Acc				
	Rnd	Maj	RF	NN	XG	Rnd	Maj	RF	NN	XG	Rnd	Maj	RF	NN	XG
Comme.	0.11	0.11	0.29	**0.30**	0.26	0.50	0.50	0.73	**0.76**	0.73	0.49	0.89	**0.90**	0.88	0.74
DEVE.	0.27	0.24	**0.40**	0.35	0.35	0.50	0.50	**0.64**	0.57	0.60	0.52	**0.76**	0.75	0.75	0.66
REPO	0.31	0.27	**0.41**	0.38	0.31	0.54	0.50	0.69	**0.70**	0.59	0.53	**0.73**	0.69	0.69	0.59
XD	0.20	0.15	**0.40**	0.34	0.30	0.57	0.50	**0.76**	0.74	0.71	0.57	**0.85**	0.84	**0.85**	0.61
DM	0.12	0.12	**0.26**	0.24	0.22	0.52	0.50	**0.72**	0.70	0.69	0.52	**0.88**	**0.88**	**0.88**	0.86

Table 8 Accuracy, AUC ROC, and AUC PRC results for the 20-instability prediction task

Project	AUC PRC					AUC ROC					Acc				
	Rnd	Maj	RF	NN	XG	Rnd	Maj	RF	NN	XG	Rnd	Maj	RF	NN	XG
Comme.	0.05	0.05	**0.31**	0.24	0.27	0.50	0.50	**0.87**	0.86	0.84	0.52	**0.95**	**0.95**	0.94	0.87
DEVE.	0.15	0.17	**0.34**	0.31	0.29	0.46	0.50	0.67	0.58	**0.68**	0.46	**0.83**	0.80	**0.83**	0.75
REPO	0.14	0.13	0.16	**0.22**	0.14	0.45	0.50	0.58	**0.71**	0.53	0.55	**0.87**	0.84	0.83	0.68
XD	0.09	0.09	**0.23**	0.14	0.19	0.45	0.50	**0.77**	0.61	0.68	0.49	**0.91**	0.90	**0.91**	0.89
DM	0.10	0.08	**0.23**	0.22	0.16	0.51	0.50	**0.71**	0.70	0.68	0.49	**0.92**	**0.92**	0.91	0.90

examples in the data, and when the area under the curve is 1, it means a perfect model. For more details about these metrics, see Davis and Goadrich [15].

5.3 Prediction results

Tables 7 and 8 present the Accuracy, AUC ROC, and AUC PRC results of the 5-instability and 20-instability prediction models, respectively, created with the three learning algorithms we experimented with (NN, RF, and XGBoost) and the two baselines (Rnd and Maj). We highlighted in bold the result of the best algorithm in every project and metric. For completeness, "Appendix 2" lists the full confusion matrices obtained by the evaluated prediction models.

Consider first the results for the 5-instability prediction models (Table 7). As can be seen, the AUC PRC and AUC ROC scores for our prediction models are significantly better than the baselines, outperforming them by more than 10% in most cases. The models generated by Random Forest (RF) achieved the best results in the open-source projects, while the Neural Network (NN) models achieved the best results for the commercial project.

In terms of Accuracy, our prediction models outperform the random prediction model (Rnd) but the majority-based prediction model (Maj) achieves better results in all projects. This is because our data are imbalanced since most of USIs are not 5-unstable. However, since our main interest is to identify unstable USIs, a classifier that never identifies an unstable USI is not useful. Furthermore, the accuracy of our prediction models is almost the same as the majority baseline in all cases.

Next, consider the results for the 20-instability prediction models (Table 8). The results are similar to the results in Table 7. The AUC PRC and AUC ROC results of the developed prediction models significantly improve over the baseline results.

Among the three learning algorithms, the models created by RF achieved the best results for most projects in terms of AUC PRC and AUC ROC. In contrast to the 5-instability results (Table 7), the best results for the commercial project were achieved by RF, while for the "REPO" project the best results were obtained by NN. Fully explaining the behavior of different supervised learning algorithms is a well-known open challenge. But, the RF classifier's overall positive performance has been noted in other high dimensional datasets and domains [9, 11]. Cases where NN outperformed RF can be explained by the neural network's ability to learn complex relations the RF model missed.

Next, we compare the results of RF when used to predict 5-instability and 20-instability (Tables 7 and 8 , respectively). It is evident that the average AUC PRC is lower and the average Acc is higher when predicting 20-instability. We conjecture that this is because there are fewer 20-unstable USIs than 5-unstable USIs. Thus, the data are more imbalanced. Achieving high Acc in an imbalanced dataset is easy for binary classification by using a simple majority rule. On the other hand, achieving high AUC PRC becomes more difficult as there are fewer instances of the minority class.

Another important observation is that there is no clear relation between the ratio of USIs that follow the standard Connextra form in a project and the performance of our prediction models on that project. The commercial, DEVELOPER, and DM projects have significantly fewer

template-based USIs but the results of our prediction models were similar across all projects.

5.4 Features groups effect

The results in Table 6 indicate that each project has a different combination of features that performed best. However, in all cases using the Personalized Metrics features proved to be beneficial, and the Process-based Metrics features were beneficial in all open-source projects. Interestingly, the advanced text features, which employ sophisticated NLP techniques, did not prove to be beneficial except for the commercial and DM projects. Note that the DM project requires all feature groups to reach the best results. This can be related to the fact that the prediction results for this project are the lowest, which indicates that the models could correctly identify unstable USIs only to a limited extent. Thus, every feature that we added improved the results, and every feature that we removed negatively affected the results. Examining the USIs in this project, we notice that they are written in a polite and unofficial language, like using the words "please", "nice" and "great" in the text. For example: *Please install v14.0 in the shared stacks on lsst-dev. The stack release v14.0 just came out. It would be useful if v14.0 is available in the shared stack on lsst-dev.* This might restrict the models' ability to classify the USI instability to a satisfactory level.

To gain a deeper insight into how each feature group affects each project, we conducted an additional experiment in which we trained a classifier for each project using features from a single feature group. In all cases, the results achieved by selecting all the groups together outperformed the results achieved when selecting each of the feature groups separately. When comparing the results of each feature group separately, the feature groups that achieved the highest performance for each project are the following:

- Commercial—simple text-based features.
- DEVELOPER—simple text-based features.
- REPO—process-based metrics.
- XD—process-based metrics.
- DM—personalized metrics.

These findings are aligned with the results shown in Table 6, where features from the most effective feature group for each project are also features that are important for the best prediction model trained for that project.

For example, text-based features are important for the commercial and the DEVELOPER projects, whereas the important features in REPO, XD, and DM include the process-based and personalized metrics. Referring to the DM project, the most important feature group is the personalized metrics. This may be because the text in this project is written less formally and therefore has a lower impact on the instability.

5.5 Summary of experimental results

Our experimental results show that our USI instability classifiers outperform baseline results, supporting our first hypothesis (H1) and answering our first research question (RQ1) affirmatively.

The answer to our second research question (RQ2) is more involved. Our feature importance analysis shows that there is a difference between the feature groups that affect the open-source and the commercial projects, where the process-based metrics affect the open-source projects and the advanced text-based metrics affect the commercial project.

These results may be because the teams in the commercial project work tightly together, including daily Scrum meetings. Thus, they had a limited incentive to update USI information via Jira. In contrast, open-source teams usually work in a more distributed manner, which makes updating USI information via Jira a more effective form of communication. Therefore, the process-based features that passed through the Jira had a higher impact on the USI instability in the open-source projects. In the commercial project, they had limited impact, and the text and related features had a higher impact. In general, our second hypothesis (H2) is only partially supported, and in fact, some feature groups were not useful at all in some projects.

6 Threats to validity

The results of this study need to be considered in view of several threats to validity [43].

Construct validity The USIs in our dataset do not all represent user stories as defined in the literature; this means that these are not user stories in the strict sense. While we mitigated this threat to some extent by only selecting USIs labeled as "user story" or "change request", a qualitative analysis of the USIs may provide additional insight into our results. The selection of the features is also a threat to validity, as there is no guarantee that these are most relevant for predicting USI instability. To overcome this threat, we consulted with experts and studied the factors that affect USI instability from the data and the development process.

Internal validity The acute imbalance of our dataset poses also a threat to the validity of this work. The percentage of unstable USIs are between 5 to 32 in our dataset, which comprises USIs from real-world projects. Imbalanced datasets are a challenge for training and for testing, as they may represent incorrectly the actual performance of the prediction models on datasets with different distributions. To deal with

this threat, we used an evaluation measure that fit imbalance data—the area under the precision-recall curve.

The choice of k—the amount of instability we aim to predict—is also a potential threat to the validity of this work. To mitigate this threat, we experimented with a range of k values. But, we mostly present results for $k = 5$ and $k = 20$ to focus our presentation. We did not observe meaningful differences in other values of k we experimented with.

Conclusion validity Another threat to validity is the choice of supervised learning algorithms we used. Indeed, other learning algorithms may yield different results. Nevertheless, the algorithms we have chosen are standard algorithms for binary classification and were effective in many other prior works. They have been used in prior work on similar topics such as software defects prediction [20], development effort estimation [1], and testing effort estimation [28].

External validity The main premise in the proposed instability metric is that in a normal workflow, user stories are not intended to be edited in a sprint in which they are implemented. While this is the common practice in the industry, other workflows may include making changes to user stories during a sprint. Our work is thus limited to projects in which this premise holds.

Another threat to validity is a selection bias of the projects in our dataset, which might affect the generalization of the results. To mitigate this threat we chose open-source projects of different sizes and from different repositories. Yet, it is required to apply and check the developed prediction models in other projects as well.

7 Conclusion and future work

In this paper, we explored a novel impact-driven approach for assessing USIs. Several metrics for assessing USIs were proposed and discussed. We focused on a specific metric called *instability*, where an unstable US is one that changes during the sprint in which it is implemented. An unstable USI suggests that it should have been described better before being assigned to a sprint. To this end, we proposed an approach based on machine learning to train a classifier that predicts whether a USI will be unstable, before it is assigned to a sprint. By identifying these USIs earlier before they enter a sprint, they could be further improved and elaborated, decreasing the chance for the wrong implementation and wasted efforts during the sprint. We evaluated our prediction models on several open-source projects and a commercial one. The results indicate that our models improved the random baseline results by more than 9% in terms of AUC PRC and AUC ROC, for all the projects. To support future research, we have made all the code and data used for this project publicly available at https://doi.org/10.5281/zenodo.5822986 and https://doi.org/10.5281/zenodo.5579864, respectively, except for the data from the commercial project, which is proprietary.[12]

Our work opens the door to several interesting directions for future work:

1. Apply our instability prediction algorithm to a larger dataset of projects which includes more commercial and open-source projects.
2. Apply more advanced NLP tools, features, and classifiers to improve the instability prediction results. For example, one may consider standard NLP features such as n-grams, and develop instability measures that consider also the semantics of the changes that have been made.
3. Apply our approach to other requirement-related artifacts, beyond user story issues. These can include use cases, conceptual models, test cases, etc.
4. Perform a cross-project analysis, in which instability prediction models trained on data from some projects is evaluated on the data from different projects.
5. Developing a method for specifying the appropriate k—the number of changes in a USI that make in unstable—that should be considered. One way to do so is by analyzing the relationship between different choices of k and other metrics that are related to productivity, e.g., the number of faults in the developed code.
6. Developing a semantics-based method to measure instability that applies NLP techniques to quantify the changes made to a USI after entering a sprint.
7. Perform a deep analysis of the types of changes observed in unstable USIs. Such an analysis may provide insights into how USIs can be written a-priori to minimize such changes.

While our instability prediction models can still be significantly improved, their relative success suggests that it can be used as a foundation of an advanced US authoring tool that will better support the management of USIs. Such a tool can be used in a workflow where a USI is evaluated automatically before entering a sprint, and the tool's output includes a recommendation about whether the evaluated USI is sufficiently detailed and ready to be implemented. Exploring the design of such a tool is yet another direction for future research.

[12] The code is written in Python and includes a detailed README file with documentation with step-by-step instructions. This code can be used to reproduce our experiments, train and evaluate instability prediction models on other datasets, and explore other features and algorithms over our dataset. Our dataset is given as an exported SQL server dump file.

Appendix 1: List of features

The features used for our instability prediction models are listed below. For each feature we briefly explain the rationale of using it and, when needed, how it is computed.

1.1: Simple text processing features

- *Text length* The number of characters in the USI text. Short text may suggest missing details, and very long text may indicate an excessively detailed USI that causes confusion.
- *Number of question marks in the text* Question marks may indicate that the USI writer is not sure about the requirement, and therefore, the probability to change increases.
- *Number of headlines in the text* In some projects, most USIs contain headlines that provide some structure to the USI text. In such cases, missing headlines can testify to missing information.

Has an acceptance criteria headline An "Acceptance Criteria" headline exists in many projects. Since the acceptance criteria are an important part of the USI, its absence may testify to missing information.

- *Number of sentences in the text* Using one long sentence can be hard to understand, as well as many short sentences.
- *Number of words in the text* Similar to the text length feature, fewer words may suggest missing details, and too many words may cause confusion.
- *Average number of words in a sentence in the text* Similar to the number of sentences in the text feature, too short/long sentences, on average, may affect the readability of the text.
- *Does the text contain a URL* This binary feature indicates whether the USI description contains a URL. A URL can add additional information that is missing in the URI text, and thus affect the USI instability.
- *Does the text contain source code* This binary feature indicates whether the USI description contains source code. As with the previous feature, the source code can add additional information that can be missing in the text.
- *Does the text contain the Connextra user story template* This binary feature indicates whether the USI description follows the well-known Connextra template ("As a (role) I want (something) so that (benefit)". We conjecture that stories that were written following this template will be of higher quality and possibly lower instability.
- *Does the text contains the words "TODO/TBD/Please"* This binary feature indicates if the USI description contain one of the words "TODO", "TBD", and "Please". These words imply something missing or unclear in the text, which may indicate that it will be edited later (exhibiting instability).
- *Number of stop-words the text contains* A sentence without stop words can be hard to understand. On the other hand, a sentence with too many stop words may suggest low writing quality and uncertainty.
- *Number of nouns/adjectives/adverbs/pronouns in the text* This feature may provide some insight into the USI description structure. For example, a USI missing a verb may be hard to understand.
- *Is text field is empty* This binary feature indicates whether the USI description was empty or not. Missing text field may suggest lack of information and thus be correlated with instability.

1.2: Advanced text-based features

- *USI Vector* Translates the documents (USIs) text into a fixed-size vector of numbers, which are the resulting feature. See more details in the main body of the text.
- *Topic Model* we train a topic model for each project and add a feature for each topic. See more details in the main body of the text.

1.3: Process metrics

.

- *Number of changes in the text before entering a sprint* Many changes in the USI before the sprint can point on a problem with the text or an unclear requirement and can cause more changes during the sprint.
- *Number of comments before entering a sprint* The number of comments this USI had before entering the sprint. If a USI has many comments, it may suggest an unclear requirement.
- *Number of changes in story point before entering a sprint* Change in the number of story points may indicate an unclear requirement and hence lead to USI instability.
- *Number of story points when entering a sprint* The estimated effort of the development team may provide indirect influence on the USI instability.
- *USI priority when entering a sprint* The priority may influence stability, e.g., if the USI is urgent, it may be written in a hasty and less rigorous manner.
- *Time until the USI entered a sprint* The rationale for this feature is that if a USI stays long in the backlog it had more time to be written in a full and stable manner. On the other hand, this may suggest that this USI

Table 9 The TN, FP, FN, TP, and accuracy results for all the projects and prediction models in our evaluation

Proj.	Alg.	$k = 5$					$k = 20$				
		TN	FP	FN	TP	Acc.	TN	FP	FN	TP	Acc.
Comm.	Maj	330	0	40	0	0.89	353	0	17	0	0.95
	NN	315	15	28	12	0.88	346	7	14	3	0.94
	RF	326	4	34	6	0.90	348	5	13	4	0.95
	Rnd	162	168	19	21	0.49	182	171	7	10	0.52
	XG	250	80	16	24	0.74	311	42	5	12	0.87
DEVE.	Maj	186	0	60	0	0.76	203	0	43	0	0.83
	NN	176	10	51	9	0.75	199	4	38	5	0.83
	RF	173	13	48	12	0.75	189	14	36	7	0.80
	Rnd	99	87	31	29	0.52	91	112	21	22	0.46
	XG	130	56	28	32	0.66	169	34	27	16	0.75
DM	Maj	875	0	117	0	0.88	909	0	83	0	0.92
	NN	871	4	116	1	0.88	904	5	81	2	0.91
	RF	875	0	117	0	0.88	909	0	83	0	0.92
	Rnd	450	425	56	61	0.52	435	474	34	49	0.49
	XG	849	26	108	9	0.86	891	18	79	4	0.90
REPO	Maj	132	0	48	0	0.73	156	0	24	0	0.87
	NN	121	11	44	4	0.69	154	2	24	0	0.86
	RF	121	11	44	4	0.69	151	5	24	0	0.84
	Rnd	72	60	24	24	0.53	89	67	14	10	0.55
	XG	86	46	27	21	0.59	118	38	20	4	0.68
XD	Maj	297	0	51	0	0.85	317	0	31	0	0.91
	NN	297	0	51	0	0.85	316	1	31	0	0.91
	RF	276	21	33	18	0.84	311	6	29	2	0.90
	Rnd	168	129	22	29	0.57	157	160	16	15	0.49
	XG	182	115	19	32	0.61	309	8	29	2	0.89

is of less importance, and hence may have been written in a sloppy manner, and eventually exhibit instability.
- *The number of issue links* The number of issue links from several USI types (as duplicate and block). A high number of dependencies may affect the USI instability.

1.4: Personalized metrics

- *Number of USIs* The number of USIs that the author wrote before the current USI. The idea behind that feature is that we expect a person with limited experience in writing USI, to write limited quality USI. On the other hand, we expect a skilled person to write high-quality USI, with a lower probability that the USI will be changed.
- *Ratio of unstable USIs in the past* The ratio of unstable USIs from all the USIs that the author wrote before the specific USI. We expect that the probability that a USI will be changed to increase when the ratio of unstable USI to a writer is high.

Appendix 2: Confusion matrices for instability prediction models

For completeness, Table 9 provides the TN, FP, FN, TP, and accuracy results obtained by the evaluated prediction models in each project. We show here results the 5-instability and 20-instability prediction tasks in the columns $k = 5$ and $k = 20$, respectively.

Acknowledgements This research was supported by the Ministry of Science & Technology, Israel, and by the Israeli Science Foundation Grant #210/17 to Roni Stern.

References

1. Abdelali Z, Mustapha H, Abdelwahed N (2019) Investigating the use of random forest in software effort estimation. Procedia Comput Sci 148(2019):343–352
2. Abrahamsson P, Fronza I, Moser R, Vlasenko J, Pedrycz W (2011) Predicting development effort from user stories. In: International symposium on empirical software engineering and measurement, pp 400–403

3. Abrahamsson P, Oza N, Siponen MT (2010) Agile software development methods: a comparative review. In: Dingsøyr T, Dybå T, Moe N (eds) Agile software development. Springer, Berlin
4. Abrahamsson P, Salo O, Ronkainen J, Warsta J (2017) Agile software development methods: review and analysis. Preprint arXiv:1709.08439
5. Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R et al. (2010) Manifesto for agile software development
6. Bishop CM (1995) Neural networks for pattern recognition. Oxford University Press
7. Breiman L (2001) Random forests. Mach Learn 45:5–32
8. Buglione L, Abran A (2013) Improving the user story agile technique using the invest criteria. In: International workshop on software measurement and international conference on software process and product measurement. IEEE, pp 49–53
9. Caruana R, Karampatziakis N, Yessenalina A (2008) An empirical evaluation of supervised learning in high dimensions. In: International conference on Machine learning. ACM, pp 96–103
10. Chen T, Guestrin C (2016) Xgboost: A scalable tree boosting system. In: ACM sigkdd international conference on knowledge discovery and data mining, pp 785–794
11. Chen X, Ishwaran H (2012) Random forests for genomic data analysis. Genomics 99:323–329
12. Choetkiertikul M, Dam HK, Tran T, Ghose A (2017) Predicting the delay of issues with due dates in software projects. Empir Softw Eng 22:1223–1263
13. Choetkiertikul M, Dam HK, Tran T, Pham TTM, Ghose A, Menzies T (2018) A deep learning model for estimating story points. IEEE Trans Softw Eng 45(7):637–656
14. Coelho E, Anirban B (2012) Effort estimation in agile software development using story points. Int J Appl Inf Syst (IJAIS) 3(7):7–10
15. Davis J, Goadrich M (2006) The relationship between Precision-Recall and ROC curves. In: The international conference on machine learning (ICML), pp 233–240
16. Dimitrijević S, Jovanović J, Devedžić V (2015) A comparative study of software tools for user story management. Inf Softw Technol 57:352–368
17. Dargan JL, Wasek JS, Campos-Nanez E (2016) Systems performance prediction using requirements quality attributes classification. Requir Eng 21:553–572
18. Femmer H, Fernández DM, Wagner S, Eder S (2017) Rapid quality assurance with requirements smells. J Syst Softw 123:190–213
19. Fowler M, Highsmith J (2001) The agile manifesto, Software Development
20. Gupta A, Shilpa S, Goyal S, Rashid M (2020) Novel XGBoost tuned machine learning model for software bug prediction. In: The international conference on intelligent engineering and management (ICIEM), pp 376–380
21. Haugen NC (2006) An empirical study of using planning poker for user story estimation. In Agile 06:23–34
22. Hayes JH, Li W, Yu T, Han X, Hays M, Woodson C (2015) Measuring requirement quality to predict testability. In: IEEE international workshop on artificial intelligence for requirements engineering (AIRE), pp 1–8
23. Hearty P, Fenton N, Marquez D, Neil M (2008) Predicting project velocity in xp using a learning dynamic bayesian network model. IEEE Trans Softw Eng 35:124–137
24. Kassab M (2015) The changing landscape of requirements engineering practices over the past decade. In: International workshop on empirical requirements engineering (EmpiRE), pp 1–8
25. Lai ST (2017) A user story quality measurement model for reducing agile software development risk. Int J Softw Eng Appl 8:75–86
26. Le Q, Mikolov T (2014) Distributed representations of sentences and documents. In: International conference on machine learning, pp 1188–1196
27. Leffingwell D (2010) Agile software requirements: lean requirements practices for teams, programs, and the enterprise. Addison-Wesley Professional
28. López-Martín C (2021) Machine learning techniques for software testing effort prediction. Softw Qual J 2021:1–36
29. Lucassen G, Dalpiaz F,van der Werf JMEM, Brinkkemper S (2015) Forging high-quality user stories: towards a discipline for agile requirements. In: IEEE international requirements engineering conference, pp 126–135
30. Lucassen G, Dalpiaz F, van der Werf JMEM, Brinkkemper S (2016) The use and effectiveness of user stories in practice. Foundation for software quality. In: International working conference on requirements engineering, pp 205–222
31. Lucassen G, Dalpiaz F, van der Werf JMEM, Brinkkemper S (2016) Improving agile requirements: the quality user story framework and tool. Requir Eng Springer 21:383–403
32. Lucassen G, Dalpiaz F,van der Werf JMEM, Brinkkemper S (2017) Improving user story practice with the Grimm Method: a multiple case study in the software industry. In: International working conference on requirements engineering: foundation for software quality. Springer, pp 235–252
33. Mahnič V, Hovelja T (2012) On using planning poker for estimating user stories. J Syst Softw 85:2086–2095
34. Paetsch F, Eberlein A, Maurer F (2003) Requirements engineering and agile software development. In: IEEE international workshops on enabling technologies: infrastructure for collaborative enterprises, pp 308–313
35. Palomares C, Franch X, Quer C, Chatzipetrou P, López L, Gorschek T (2021) The state-of-practice in requirements elicitation: an extended interview study at 12 companies. Requir Eng 26(2):273–299
36. Porru S, Murgia A, Demeyer S, Marchesi M, Tonelli R (2016) Estimating story points from issue reports. In: International conference on predictive models and data analytics in software engineering. ACM, pp 1–10
37. Rees MJ (2002) A feasible user story tool for agile software development? In: Proceedings of the ninth asia-pacific software engineering conference, pp 22–30
38. Schwaber K, Beedle M (2002) Agile software development with Scrum. Prentice Hall, Upper Saddle River
39. Schwaber K, Sutherland J (2020) The scrum guide: the definitive guide to scrum: the rules of the game. http://www.scrum.org/scrum-guides. Accessed 31 Dec 2021
40. Shi L, Wang Q, Li M (2013) Learning from evolution history to predict future requirement changes. In: IEEE international requirements engineering conference (RE), pp 135–144
41. Wallach HM (2006) Topic modeling, beyond bag-of-words. In: The international conference on machine learning (ICML), pp 977–984
42. Wang X, Zhao L, Wang Y, Sun, J (2014) The role of requirements engineering practices in agile development: an empirical study. Requirements Engineering. Springer, pp 195–209
43. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer
44. Zhao L, Waad A, Ferrari A, Letsholo KJ, Ajagbe MA, Chioasca EV, Batista-Navarro RT (2021) Natural language processing for requirements engineering: a systematic mapping study. ACM Comput Surv 54(3):1–41
45. Ziauddin SKTZS (2012) An effort estimation model for agile software development. Adv Comput Sci Appl (ACSA) 2:314–324