

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/359427435>

# Towards Reliable Agile Iterative Planning via Predicting Documentation Changes of Work Items

Conference Paper · March 2022

DOI: 10.1145/3524842.3527943

CITATIONS

0

READS

172

3 authors, including:



[Jirat Pasuksmit](#)

University of Melbourne

6 PUBLICATIONS 6 CITATIONS

[SEE PROFILE](#)



[Patanamon Thongtanunam](#)

University of Melbourne

35 PUBLICATIONS 705 CITATIONS

[SEE PROFILE](#)

# Towards Reliable Agile Iterative Planning via Predicting Documentation Changes of Work Items

Jirat Pasuksmit  
jpasuksmit@student.unimelb.edu.au  
The University of Melbourne  
Australia

Patanamon Thongtanunam  
patanamont@unimelb.edu.au  
The University of Melbourne  
Australia

Shanika Karunasekera  
karus@unimelb.edu.au  
The University of Melbourne  
Australia

## ABSTRACT

In agile iterative development, an agile team needs to analyze documented information for effort estimation and sprint planning. While documentation can be changed, the documentation changes after sprint planning may invalidate the estimated effort and sprint plan. Hence, to help the team be aware of the potential documentation changes, we developed DocWARN to estimate the probability that a work item will have documentation changes. We developed three variations of DocWARN, which are based on the characteristics extracted from the work items (DocWARN-C), the natural language text (DocWARN-T), and both inputs (DocWARN-H).

Based on nine open-source projects that work in sprints and actively maintain documentation, DocWARN can predict the documentation changes with an average AUC of 0.75 and an average F1-Score of 0.36, which are significantly higher than the baselines. We also found that the most influential characteristics of a work item for determining the future documentation changes are the past tendency of the developers and the length of description text. Based on the qualitative assessment, we found that 40%-68% of the correctly predicted documentation changes were related to scope modification. With the prediction of DocWARN, the team will be better aware of the potential documentation changes during sprint planning, allowing the team to manage the uncertainty and reduce the risk of unreliable effort estimation and sprint planning.

## ACM Reference Format:

Jirat Pasuksmit, Patanamon Thongtanunam, and Shanika Karunasekera. 2022. Towards Reliable Agile Iterative Planning via Predicting Documentation Changes of Work Items. In *19th International Conference on Mining Software Repositories (MSR '22)*, May 23–24, 2022, Pittsburgh, PA, USA. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3524842.3527943>

## 1 INTRODUCTION

Agile iterative development is a software development process where an agile team implements and delivers software increments in short iterations (e.g., “sprint” in Scrum). As a sprint is time-boxed, the team needs to accurately plan what the team will deliver after the sprint [44]. However, the correct identification of the scope of the next sprint is challenging. To acquire confidence that

the team has made a reasonable commitment, the team analyzes the documented information to estimate effort (e.g., story points) for each work item and selects the work items to work in the sprint [19, 35, 38, 44]. Hence, planning the sprint based on stable and reliable documented information would enable the team to achieve reliable sprint planning [8, 44].

Based on the Agile manifesto that comprehensive documentation has lower priority than working software [14], it is possible that the team may not foster adequate or up-to-date information in documentation when it is needed for effort estimation and sprint planning [19, 25, 38, 51]. Indeed, a recent survey reported that most of the participated agile developers considered that documented information is important when estimating effort and re-estimation would be performed when the documented information was changed [38]. Hence, it is possible that the documentation changes could invalidate the original understanding of the work item, which in turn will negatively impact the effort estimation accuracy and the sprint plan [33, 35, 38].

Hence, an approach that enables the team to be aware of the potential documentation changes would help the team gain confidence in effort estimation and sprint planning. While several studies proposed approaches to predict the uncertainty in several aspects (e.g., predicting the delay of work items [4, 10, 29] or predicting effort for requirement changes [2, 50]), none of the prior work investigated techniques to predict the documentation changes.

In this paper, we are the first to develop an approach, called DocWARN, to predict documentation changes (i.e., whether the documented information of a work item will be changed after sprint planning). With the prediction of DocWARN, the team will be able to manage the uncertainty and reduce the risk of unreliable effort estimation and sprint planning (see motivating scenarios in Section 2, which illustrate the usefulness of DocWARN in real-world practices). We developed three variations of DocWARN based on (1) the characteristics of work items (DocWARN-C), (2) the natural language text (DocWARN-T), and (3) both inputs (DocWARN-H). We built DocWARN-C using a machine learning technique (i.e., Random Forest) and built DocWARN-T and DocWARN-H using a deep learning technique with a context-sensitive word embedding method (i.e., DistilRoBERTa). To evaluate our approaches, we addressed the three following research questions:

### (RQ1) How well can we predict documentation changes?

**Motivation:** The ability to predict future documentation changes would help an agile team to be aware of and manage the potential uncertainty, reducing the risk of unreliable effort estimation and sprint plan. In this work, we proposed a prediction model called DocWARN to estimate the probability that a work item will have documentation changes after it was assigned to a sprint. Hence, we

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

MSR '22, May 23–24, 2022, Pittsburgh, PA, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9303-4/22/05...\$15.00

<https://doi.org/10.1145/3524842.3527943>

set up this RQ to evaluate the accuracy of our approaches, compared against two baselines (i.e., random guessing and OneR [59]).

**Result:** We found that the three variations of DocWARN can predict the documentation changes significantly better than the baselines. Moreover, we found that DocWARN-C achieves the best performance among the three variations with an average AUC of 0.75 and an average F1-Score of 0.36, which is 36% and 80% higher than random guessing and OneR (in terms of F1-Score), respectively. These results of our approach suggest that the characteristics of work items can be used to predict whether the documented information of a work item will be changed after the sprint planning.

**(RQ2) What are the most influential characteristics of a work item for determining the documentation changes?**

**Motivation:** In addition to predicting documentation changes, the characteristics that influence the prediction of DocWARN would provide an insight into the characteristics of the work items that are likely to have documentation changes. Therefore, the team can pay attention to a set of work items with such characteristics to improve the documentation quality, which will lead to more reliable effort estimation and sprint planning. In this study, we set out to examine the influential characteristics of the work items in DocWARN.

**Result:** We found that the past tendency of developers and the number of word count in the description text at the sprint assignment time were the most influential metrics of work items to determine the future documentation changes. This result suggests that the work items with these influential characteristics may need attention as they might have documentation changes in the future.

**(RQ3) What are the reasons for the documentation changes in the work items that we can correctly predict?**

**Motivation:** The documentation changes may occur due to various reasons. It would be beneficial to predict the documentation changes that can impact the estimated effort and the sprint plan. In other words, identifying only trivial documentation changes may unnecessarily increase effort and concern to the team when performing effort estimation and sprint planning. Hence, we perform a qualitative assessment on the documentation changes that DocWARN can predict. In particular, we manually examine the reasons that could lead to the documentation changes.

**Result:** We found that the reasons for the documentation changes that DocWARN-C can predict were related to changing scope (27%-47%), defining scope (6%-27%), adding implementation detail (4%-20%), and adding additional detail (0%-30%). This result suggests that the majority of documentation changes that DocWARN can identify are related to scope modification, i.e., changing scope and defining scope.

**Significance & Contributions:** Our findings lead us to conclude that DocWARN can predict the work items that will have documentation changes related to scope modification. As prior work found that documentation changes related to modifying scope can have an impact on the estimated effort and sprint plan [35], it implies that DocWARN can predict significant documentation changes. In addition, the influential characteristics that we found can help the team pay attention on and better manage the work items that are likely to have future documentation changes, e.g., the team can include the developers who are familiar with the context of the work items in the sprint planning. Thus, we believe that our DocWARN

and our findings can help the team reduce the risk of unstable effort estimation and sprint planning.

**Open Science:** To facilitate future work, we provide the replication package which contains the script to build DocWARN, the datasets, the fine-tuned DistilRoBERTa used for DocWARN-T and DocWARN-H, the description of how we extract the metrics for DocWARN-C, and the results of each research question in the replication package of our work.<sup>1</sup>

**Novelty:** This paper is the first to:

- Present DocWARN – an approach to predict the future documentation changes for Agile iterative development.
- Present three variations of the approach, i.e., based on (1) the characteristics of work items (DocWARN-C), (2) description text (DocWARN-T), and (3) both inputs (DocWARN-H).
- Provide the DistilRoBERTa text representation model that is fine-tuned based on the description text of 119,254 work items used in DocWARN-T and DocWARN-H.
- Demonstrate that, among the three DocWARN variations, using the characteristics of work items (DocWARN-C) achieved the best performance in predicting future documentation changes.
- Demonstrate that, the past tendency of developer and the length of the description text are the most influential characteristics of a work item in predicting the documentation changes.
- Demonstrate that, the documentation changes that are related to modifying the scope of work can be predicted by DocWARN.

## 2 BACKGROUND AND MOTIVATION

An Agile software development team works in time-boxed iterations, which are commonly referred to as “sprints”. At the beginning of each sprint, the team performs sprint planning. They estimate the effort required to implement each work item based on the available information. Then, they select a set of work items to be implemented in the sprint based on the estimated effort in conjunction with other factors (e.g., developers’ workload and expertise) [7, 44]. This set of work items will be committed to deliver at the end of the sprint. Hence, reliable effort estimation helps the team create a reliable sprint delivery [8, p.7].

To acquire confidence that the team has made a reasonable commitment for a sprint, the team analyzes the available information that is recorded in the documentation (i.e., documented information) to understand the work to be done for each work item. The information can be documented in various forms, e.g., user stories, functional requirements, use cases, acceptance criteria. A recent survey study also reported that 97% of the participated agile developers used documentation to assist effort estimation and considered that documented information is moderately to extremely important when estimating effort [38]. However, due to the nature of agile, the documented information may be changed even after effort estimation is finalized (i.e., after the sprint planning is finished) [19, 35] and re-estimation is often performed when the

<sup>1</sup>The replication package is available at <https://github.com/awsm-research/docwarn-replication>

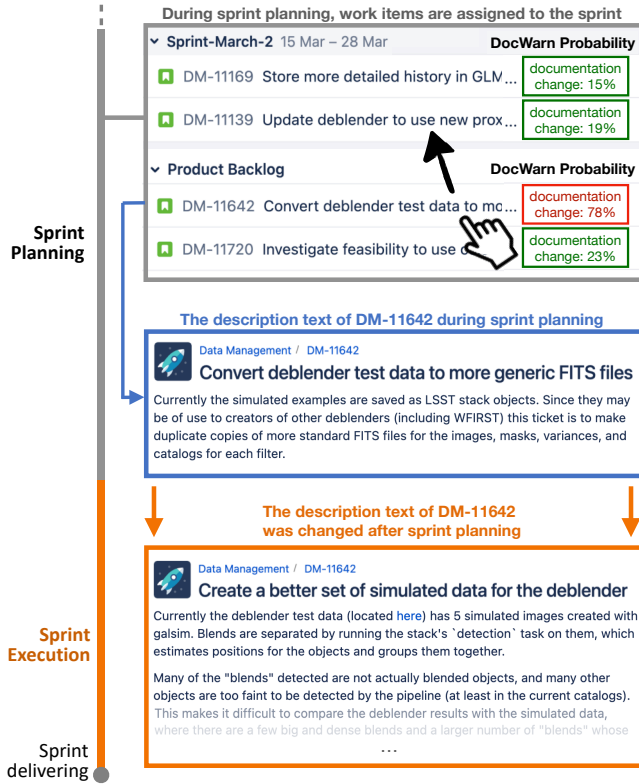


**Figure 1: Motivating Example: An example of work items selection during sprint planning (gray box), a documentation change after sprint assignment time (blue and orange boxes), and the integration of DocWARN (red and green boxes).**

documented information is changed [38]. Intuitively, the documentation change could indicate that the work item is not stable [5], which might lead to a misunderstanding during effort estimation and sprint planning. In particular, if the documentation change is significant, the estimated effort and the sprint plan may become inaccurate and more effort will then be required for re-estimation and re-planning [33, 35, 38, 44, p.339]. Otherwise, the sprint delivery will be delayed or the team will have to absorb the higher workload by themselves [38]. Hence, we propose **DocWARN**, an approach to predict future documentation changes, to help the team be aware of the potential change in the documented information before it will be used for effort estimation and sprint planning.

## 2.1 Motivating Scenarios.

To illustrate the usefulness of DocWARN in real-world practices, we provide two motivating scenarios with an actual example of the documentation change. Consider an agile team planning a sprint by selecting work items from the product backlog (see Figure 1). Based on the sprint goal, estimated effort and other factors, these four work items are the candidates that can be developed in the sprint. There could be two possible scenarios:

*Without DocWARN*, the team may select a work item without being aware that the documented information will be changed.

Then, in this scenario, all four work items are selected for this sprint. However, one of these work items will eventually have a documentation change after the sprint plan (i.e., DM-11642). Figure 1 shows a real-world example of DM-11642.<sup>2</sup> When DM-11642 was assigned to the sprint (see the gray box), the documented information describes that the work item was aimed to “Convert debundler test data to more generic FITS files” (see the blue box). After that, the documented information was changed as per the changed scope of work (see orange box). The team also noted that “This is a change in scope from the original description of this ticket, which was just to make the current images more accessible to non-LSST Stack versions of the debundler”. Consequently, the estimated effort was increased from 1 to 10 which may exceed the team capacity, causing the delayed delivery (i.e., the work item was postponed to the next sprint).

*With DocWARN*, DocWARN will provide the probability that the documented information of each work item will be changed (see the green and red boxes in Figure 1). This will allow the team to be aware of the potential uncertainty and manage this in several ways. For example, to reduce the risk of unreliable effort estimation and sprint planning, the team may not select the work item with a high probability of documentation change (i.e., DM-11642) to work in the sprint. Then, the team can later progressively perform detail analysis on DM-11642 until they are confident that the details of documented information are sufficient.

Aligning with the just-in-time practice of agile, DocWARN does not mandate the team to spend additional effort to perform comprehensive documentation at the early development stage. Instead, the team can use the recommendation of DocWARN to support them at any time when documented information is used. For example, as illustrated in our motivating scenario, DocWARN can be used to support the decisions during effort estimation and sprint planning.

## 3 DOCWARN: PREDICTING DOCUMENTATION CHANGES

The documentation changes were often found along with re-estimation, which may negatively impact the sprint plan [35, 38]. To help agile teams be aware of the potential documentation changes after sprint planning, we developed DocWARN to estimate the probability that a work item will have documentation changes. Since the documentation changes can be trivial (e.g., typo fixing, clarification) [33, 44], we consider that a work item had a documentation change if its description text (i.e., the summary and description) was semantically changed more than 10%<sup>3</sup> after it was assigned to a sprint. Prior studies found that the metrics and description text of work items can be used to predict the delivery delay [4] and the effort [6, 40, 47]. Inspired by prior studies, we developed three variations of DocWARN. We developed DocWARN-C based on the characteristics of a work item, developed DocWARN-T based on description text of work items, and developed DocWARN-H based on both inputs. In this section, we describe the approaches to build the three variations of DocWARN (also shown in Figure 2).

<sup>2</sup><https://jira.lsstcorp.org/browse/DM-11642>

<sup>3</sup>We discussed this threshold in Section 7.1.

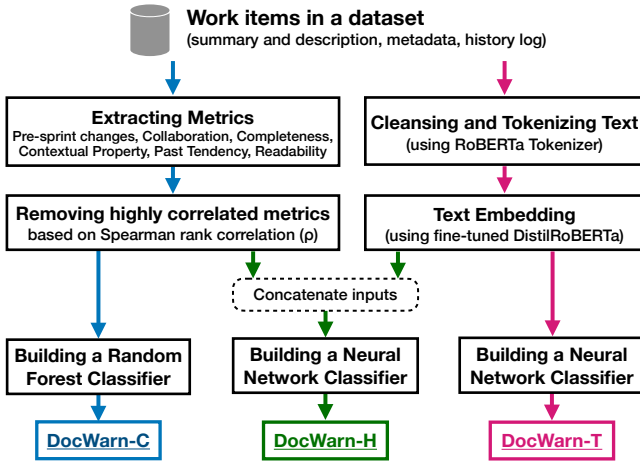


Figure 2: Overview of the approach to build DocWARN-C, DocWARN-T, and DocWARN-H.

### 3.1 DocWARN-C (characteristics)

The characteristics of the work items (e.g., types, priority, number of changes) were shown to be effective in predicting the delay to deliver [4] and the effort [40, 47] of work items. It is possible that these characteristics can also be used to predict documentation changes. Hence, we built DocWARN-C, which predicts the documentation changes based on the characteristics when the work items were assigned to a sprint. We used a Random Forest classifier to build the DocWARN-C [31]. Figure 2 (blue line) shows three steps of building DocWARN-C, which we describe in detail below.

**Extracting Characteristics:** We used 41 metrics to capture the characteristics of a work item, which can be organized into six dimensions. Table 1 provides descriptions of our metrics. These characteristics were extracted at the time when a work item was assigned to a sprint. The intuitions of our metrics for each dimension are as follows. For the pre-sprint changes metrics, we hypothesized that a work item whose documented information is frequently changed before sprint assignment time might also have documentation changes after it is assigned to a sprint [5]. Our intuition for the collaboration metrics was that the developer who frequently worked with many people might provide stable information. For the completeness dimension, we hypothesized that the work item with complete information should be less likely to be changed. It is possible that the documentation of a work item may be changed in specific contexts (e.g., work item type). Hence, we considered the primitive attributes which provide contextual information of a work item [4, 5, 40]. For the past tendency dimension, it is possible that the work items that were reported by or assigned to experienced developers may be more stable [35]. Lastly, for the readability dimension, we hypothesized that work items that are easier to understand may be less likely to be changed. Noted that we describe how we extract our metrics in the replication package.<sup>1</sup>

**Removing Correlated Metrics:** Prior work reported that a set of highly correlated metrics might mislead the classifiers [53]. Hence,

we performed the Spearman rank correlation ( $\rho$ ) analysis and removed the highly correlated metrics, i.e.,  $|\rho| > 0.7$ . To avoid subjective bias and produce a consistent set of selected metrics, we used AutoSpearman [21] to perform the correlation analysis and automatically select the metrics that have the least correlation with other metrics in the training dataset.

**Building a Classifier:** We built the DocWARN-C using a Random Forest classifier, i.e., the classification technique that is known to have good overall accuracy and to be robust to outliers as well as noisy data [31].<sup>4</sup> To build the Random Forest classifier, we used the *randomForest* function from the R *randomForest* package [30].

### 3.2 DocWARN-T (text)

Aside from the characteristics of work items, prior work shows that the description text of work items can be used to predict the effort of work items [6, 40]. Therefore, it is possible that description text can also be used to predict documentation changes. We built DocWARN-T to predict the documentation changes based on the description text written in a work item at the time when it was assigned to a sprint. In this work, we used a deep learning technique with a context-sensitive word embedding method called “DistilRoBERTa” [45]. Figure 2 (pink line) shows three steps of building DocWARN-T, which we describe in detail below.

**Tokenizing Text:** The documented information, regardless of the form, is used to understand how the work item should be done. Therefore, we treated the documented information as a simple text (i.e., no specific form). Specifically, for each work item, we first concatenated all sentences in the summary and description as one description text. Then, we removed the JIRA text formatting notation<sup>5</sup> and removed special characters (e.g., @, &, \*) as these notations do not provide any information related to a work item. To reduce the complexity of the description text, we replaced text in the JIRA code block notation (i.e., {code}...{/code}) with the keyword TAG\_CODE and replaced a hyperlink with the domain name (e.g., “www.website.com/page” were replaced with “websitecom”). Then, we tokenized the description text into a sequence of tokens using the *RoBERTaTokenizerFast* library from the HuggingFace Python package [62].

**Text Embedding:** We used DistilRoBERTa [45] (i.e., a lightweight version of the state-of-the-art text representation model [32]) to embed the description text into a contextualized vector. Unlike other embedding techniques (e.g., TF-IDF, Bag of Words), DistilRoBERTa considers all tokens in the sequence when generating a vector, e.g., “my home” and “my home page” will be embedded differently. Since DistilRoBERTa is pre-trained based on OpenWebTextCorpus [15], we fine-tuned it with our natural language dataset in the software development context to improve its performance on the downstream task (i.e., predicting documentation changes). To do so, we used the description text of the work items from all studied projects (119,254 work items) to fine-tune the pre-trained DistilRoBERTa [45]. We used the Masked Language Modeling technique same as when DistilRoBERTa was pre-trained [45]. Finally,

<sup>4</sup>We also conducted an experiment with logistic regression (see Section 7).

<sup>5</sup><https://jira.atlassian.com/secure/WikiRendererHelpAction.jspa>

**Table 1: A list of metrics that we used to measure the characteristics of a work item at sprint assignment time.**

| Dimension           | Metric                                                                                                                                                                                                                             | Description                                                                                                                                                                                                                                                                                 |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pre-Sprint Changes  | textdesc-change-count, assignee-change-count, comment-count                                                                                                                                                                        | The number of change activities (i.e., changing text, changing assignee, comments) occurred <i>before</i> the work item was assigned to a sprint.                                                                                                                                           |
| Collaboration       | in-degree, out-degree, total-degree, kcoreness, clustering-coefficient, closeness-centrality, betweenness-centrality, eigenvector-centrality                                                                                       | Degree of activity that the reporter of work item recently had with other developers in his/her collaboration network in the project.                                                                                                                                                       |
| Completeness        | has-acceptance-criteria, has-code, has-attachments, has-testcases, has-stacktraces, number-bullet-tasks, number-hyperlinks                                                                                                         | Whether the description text contained the technical information.                                                                                                                                                                                                                           |
| Primitive Attribute | workitem-type, priority, components, number-watchers, number-votes, has-subtasks, has-epic                                                                                                                                         | The attributes of a work item that are directly extracted from JIRA metadata.                                                                                                                                                                                                               |
| Past Tendency       | reporter-workitem-num, reporter-recent-workitem-num, reporter-stable-docchg-rate, reporter-stable-sprintchg-rate, assignee-workitem-num, assignee-recent-workitem-num, assignee-stable-docchg-rate, assignee-stable-sprintchg-rate | The past tendency of the work items reporter and assignee in terms of the involved work items in the past, the recently involved work items, the rate of involved work items that did not have documentation changes, and the rate of involved work items that did not have sprint changes. |
| Readability         | flesch, fog, lix, kincaid, ari, coleman-lieu, smog, wordcount                                                                                                                                                                      | The scores of how difficult to understand the description text. To estimate the readability score, we used seven readability tests, i.e., Flesch [13], Fog [16], Lix [22], Flesch-Kincaid [26], Automated Readability Index [49], Coleman-Liau [9], and SMOG [36].                          |

\*Due to space limitation, we provided the description of each metrics and how we extract them in our replication package.<sup>1</sup>

we used the fine-tuned DistilRoBERTa to embed the whole tokenized description text into a contextualized vector with the size of 768 (default setting).

**Building a Classifier:** We used the embedded vectors of description texts to train a Fully Connected neural network with the Sigmoid activation function [18]. We trained the neural network model for 20 epochs with a batch size of 64, dropout ratio of 0.2, and used binary cross-entropy as the loss function. To minimize the loss function, we used the Adam optimizer with a learning rate of 0.005. These are our overall best-performing settings based on our experiment on different hyper-parameters, i.e., number of epochs of 10-50, batch size of 4-256, dropout ratio of 0.1-0.5, learning rate of 0.0001-0.1, and several loss function optimizers (e.g., Adam, RMSProp, Stochastic Gradient Descent).

### 3.3 DocWARN-H (hybrid)

The combination of the description text and the metrics that capture the characteristics of the work items may provide a better prediction of documentation changes. Hence, we built DocWARN-H to predict the documentation changes based on both the description text and the extracted metrics. Figure 2 (green line) provides an overview of building DocWARN-H. We concatenate the vector of the metrics extracted in DocWARN-C and the contextualized vector of description text from DocWARN-T into one vector. The length of the concatenated vector is 768 plus the number of metrics that passed the correlation analysis. After that, similar to DocWARN-T, we use the concatenated vector to train a Fully Connected neural network with the Sigmoid activation function [18]. We trained the neural network model with the same settings as DocWARN-T.

**Table 2: An overview of the studied projects.**

| Project                       | Period    | #work items |         |      |
|-------------------------------|-----------|-------------|---------|------|
|                               |           | Total       | Studied | Chg* |
| Apache DataLab (DATALAB)      | 2018-2021 | 1,896       | 695     | 157  |
| Data Management (DM)          | 2014-2021 | 30,649      | 6,831   | 830  |
| Apache Mesosphere (MESOS)     | 2011-2021 | 10,190      | 1,258   | 92   |
| Talend Data Prep (TDP)        | 2015-2021 | 6,060       | 1,780   | 554  |
| Talend Data Quality (TDQ)     | 2008-2021 | 15,761      | 2,858   | 833  |
| Titanium SDK/CLI (TIMOB)      | 2011-2021 | 22,446      | 1,165   | 108  |
| Talend MDM (TMDM)             | 2009-2021 | 9,115       | 553     | 80   |
| Talend Unified Platform (TUP) | 2011-2021 | 19,430      | 1,150   | 84   |
| Spring XD (XD)                | 2013-2018 | 3,707       | 1,441   | 160  |

Chg\*: work items that have documentation changes after sprint assignment time

## 4 CASE STUDY DESIGN

In this section, we present our studied projects, the data cleansing process, and the process to identify the documentation changes.

### 4.1 Studied Projects

This study aims to help the team better manage the uncertainty in sprint planning. We opted to study the projects where the developers work in sprints and use an online task management system to document work items. Therefore, we use the following criteria to select our studied projects.

**Criterion 1: Work in sprints.** Since our study aims to predict the documentation changes after sprint planning, we study the projects that actively develop the software in sprints. Specifically, we opt to study the projects that work in sprints

and have a relatively large proportion of work items that were assigned to a sprint.

**Criterion 2: Actively maintain documentation.** Some projects may not actively record and maintain their document. Instead, the team may use other methods to convey the information among the team members [27]. Hence, we selected the projects that tend to document long description text for a work item and often update the work item.

To find the studied projects, we started with the software projects of five large open-source software organizations, i.e., Apache, Appcelerator, LSSTCorp, Talendforge, and Spring.<sup>6</sup> These organizations use JIRA to manage their development tasks. We obtained 24 projects that develop in sprints (i.e., the projects have sprint backlogs in their JIRA). Then, based on the first criterion, we excluded the projects that have the proportion of work items (i.e., the issues in JIRA, except the bug reports) assigned to a sprint lower than the 10<sup>th</sup> percentile of all the 24 projects. Hence, 21 projects satisfied our first criterion. Based on the second criterion, we excluded the projects where (1) the work items had a relatively short description text and (2) the work items were not often updated. In particular, we exclude the projects that have the average word count of description text and the average number of updates in work items lower than the 10<sup>th</sup> percentile of all projects (i.e., the average word count < 32 and the average number of updates < 12). Hence, 16 projects satisfied our second criterion. In addition, after data cleansing (see Section 4.2), we excluded the projects that have less than 500 work items since the datasets are too small for model training. Finally, we obtained nine projects for our study. Table 2 lists our studied projects and shows the number of studied work items.

## 4.2 Data Cleansing

In this study, we only focused on the work items that were already implemented and delivered in a sprint. Table 2 shows the number of studied work items for each studied project. Specifically, we only studied the work items that (1) were assigned to a sprint, (2) with a status of *done*, *closed*, or *resolved*, and (3) with a resolution of *done*, *complete*, or *fixed*. In addition, we chose only the work items that focus on feature development, i.e., *story*, *improvement*, *new feature*, *enhancement*, *backlog task*, *technical task*, *work item*, *task*, and *sub-task*. We did not consider *bugs* because the work related to fixing bugs usually represents more uncertainty than the other types of work items, i.e., fixing bugs requires additional investigation and trial and error to identify the root cause of a bug. We also did not consider *epics* since the team is not directly working on them [44].

## 4.3 Identifying Documentation Changes

In this work, we considered that a work item had a documentation change if its description text (i.e., the summary and description) was semantically changed more than 10% after it was assigned to a sprint. Figure 3 shows the two example scenarios to identify documentation changes. We first retrieved the description text (i.e., the summary and description) at the sprint assignment time by reverting the description text of a work item to its latest version before the work item was assigned to a sprint based on its history

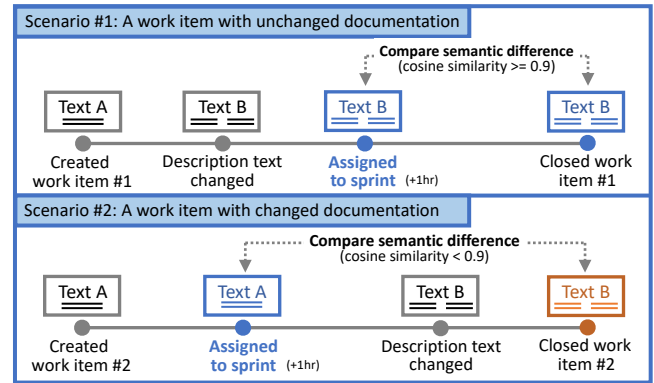


**Figure 3: Identifying documentation changes that occurred after a work item was assigned to a sprint**

log. Then, we measured the semantic difference between the description text at sprint assignment time (i.e., when the work item was assigned to a sprint) and its final version (i.e., the latest version in the history log).

We used cosine similarity to measure the semantic difference of the two versions of description text. Cosine similarity measures the similarity (angle) between two vectors of documents. Unlike the other methods (e.g., Jaccard similarity), cosine similarity measures the similarity in a multi-dimensional space (i.e., a set of words and their frequencies) [20]. In particular, we transformed the two versions of description text into two vectors of term frequencies and calculated their cosine similarity. To do so, we used *TfidfVectorizer.fit\_transform* and *cosine-similarity* functions from the Python Scikit-learn package [39]. Finally, we consider that a work item has a documentation change if the cosine similarity between the two versions of description text is lower than 0.9.

## 5 CASE STUDY RESULTS

This section describes the approaches and the results for each research question.

### 5.1 (RQ1) How well can we predict documentation changes?

**Approach:** We examined whether DocWARN can predict documentation changes by comparing the performance of DocWARN with baselines. Below, we describe how we evaluated and compared the performance of DocWARN against the two baselines, i.e., random guessing and OneR.

**Evaluating Performance:** To evaluate the performance of DocWARN, we measured the Area Under the receiver operator characteristic Curve (AUC) and F1-score. AUC measures the area under the curve that plots between the true positive rate and the false positive rate. An AUC value of 1 indicates that our model can perfectly discriminate the work items that will and will not have documentation changes. F1-score is the harmonic mean of Precision and Recall. The higher F1-score indicates that our model can better identify the work items that will have documentation changes. A prior study reported that threshold-depending measures like F1-score might

<sup>6</sup><https://issues.apache.org/jira>, <https://jira.appcelerator.org>, <https://jira.lsstcorp.org>, <https://jira.talendforge.org>, and <https://jira.spring.io>

be impacted by an imbalance dataset [54]. Therefore, instead of using a threshold of 0.5, we determined the optimal threshold based on the training dataset for each model. The optimal threshold was defined as the middle value between (1) the first quartile of the probabilities that the work items will have a documentation change ( $P_c$ ) and (2) the third quartile of the probabilities that the work items will not have documentation changes ( $P_{nc}$ ). In particular, we use a calculation of  $\frac{1}{2}(P_c + P_{nc})$ .

We performed repeated 5-fold cross-validation to validate the performance of our models [42]. For each approach and each project, we repeated the 5-fold cross-validation ten times, which produced the evaluation results for 50 models. For each round of cross-validation, we split the data of a studied project into five folds where four folds were used as a training dataset and the remaining one fold was used as a testing dataset. We used stratified random sampling to generate each fold while preserving the proportion between the work items that will and will not have documentation changes.

**Baseline Approaches:** Since none of the prior work has investigated techniques to predict documentation changes, we opt to use random guessing and OneR as our baseline approaches. For random guessing, we randomly select the target prediction class of whether the documentation of a work item will be changed (i.e., True or False). OneR is a simple supervised classification algorithm that predicts future documentation changes based on one metric that achieves the smallest error [59]. Note that AUC does not apply to random guessing and OneR classifiers because these two techniques do not estimate probability.

**Comparing Performance:** We compared the distribution of F1-score of the three DocWARN variations against the baselines. For each studied project, we performed the one-sided Wilcoxon signed-rank test [61]. We also measure the effect size ( $r$ ), i.e., the magnitude of the difference between two distributions using a calculation of  $r = \frac{Z}{\sqrt{n}}$  where  $Z$  is a statistic Z-score from the Wilcoxon signed-rank test and  $n$  is the total number of samples [55]. We interpreted the effect size based on the rule of thumb [55], i.e.,  $r \geq 0.8$  is large,  $0.5 \leq r < 0.8$  is medium,  $0.2 \leq r < 0.5$  is small, and otherwise negligible. A larger and significant difference ( $p < 0.05$ ) indicates that DocWARN outperforms the baselines.

**Result:** Figure 4 shows the distributions of performance of the model in terms of AUC, F1-score, precision, and recall of the three variations of DocWARN and the two baselines. Across all studied projects, DocWARN-C, DocWARN-T, and DocWARN-H achieved an average AUC of 0.75, 0.63, and 0.66, and the average F1-score of 0.36, 0.3, and 0.33, respectively. On the other hand, random guessing and OneR achieved an average F1-score of 0.23 and 0.07, respectively. Table 3 shows that the F1-score of the three DocWARN variations are statistically higher than the baselines with a small to large effect size ( $p < 0.01$ ) in seven projects (except MESOS/DocWARN-C and TDP/DocWARN-T). These results suggest that DocWARN tends to perform significantly better than the two baselines.

Furthermore, DocWARN-C achieved the best performance among the three variations of DocWARN with an average AUC of 0.75 and an average F1-score of 0.36 (see Figure 4). The Wilcoxon signed-rank test suggested that DocWARN-C achieved a significantly better AUC than DocWARN-T and DocWARN-H with a medium to large effect size ( $p < 0.001$ ). In terms of F1-score, DocWARN-C performed

**Table 3: A comparative summary of the performance (F1-scores) between the three variations of DocWARN and the two baselines.**

| Project | DocWARN-C      |      | DocWARN-T |                | DocWARN-H |      |
|---------|----------------|------|-----------|----------------|-----------|------|
|         | Random         | OneR | Random    | OneR           | Random    | OneR |
| DATALAB | M***           | L*** | S**       | L***           | M***      | L*** |
| DM      | L***           | L*** | L***      | L***           | L***      | L*** |
| MESOS   | N <sup>o</sup> | L*** | L***      | L***           | L***      | L*** |
| TDP     | L***           | L*** | M***      | N <sup>o</sup> | L***      | M*** |
| TDQ     | L***           | L*** | L***      | L***           | L***      | L*** |
| TIMOB   | L***           | L*** | L***      | L***           | L***      | L*** |
| TMDM    | M***           | L*** | S***      | L***           | S***      | L*** |
| TUP     | M***           | L*** | M***      | L***           | M***      | L*** |
| XD      | L***           | L*** | L***      | L***           | L***      | L*** |

Statistical significance: \*\*\*  $p < 0.001$ , \*\*  $p < 0.01$ , \*  $p < 0.05$ , <sup>o</sup>  $p \geq 0.05$

significantly better than DocWARN-T in eight projects and significantly better than DocWARN-H in seven projects with a small to large effect size ( $p < 0.05$ ). These results suggest that characteristics of a work item are a better predictor of documentation changes than textual information.

Given that the studied data is imbalanced (see Table 2), we believe that our DocWARN achieved a reasonably good accuracy. In other words, since the majority of work items in the datasets are in a FALSE class (i.e., an average of 84% of work items do not have documentation changes), it is very challenging to precisely identify only the ones with documentation changes. Nevertheless, with this accuracy of DocWARN-C, it is still valuable to practitioners to pay attention to a subset (not all) of work items to confirm if their work items are ready.

**Findings:** The three variations of DocWARN can predict the documentation changes significantly better than the baselines. DocWARN-C (the best variation of DocWARN) can predict documentation changes after the work items were assigned to a sprint with an average AUC of 0.75 and an average F1-score of 0.36.

**Implication:** Our approach based on the characteristics of work items can predict the documentation changes that occurred after sprint assignment time.

## 5.2 (RQ2) What are the most influential characteristics of a work item for determining the documentation changes?

**Approach:** Our RQ1 showed that among the three variations of DocWARN, DocWARN-C achieved the best performance to predict the documentation changes. Hence, it would be beneficial to examine the influence of the characteristics (metrics) of the work items with documentation changes. Since DocWARN-C achieved the best performance, we examine the influential metrics in DocWARN-C. Below, we describe the detail of the processes.

**Measuring Metric Influence:** We measured the influence of each metric in DocWARN-C based on the mean decrease accuracy (MDA). MDA is estimated based on the randomly permuted values of a metric [3, 31]. The larger the MDA value is, the more influence the metric has on the model. To estimate the MDA, we used the

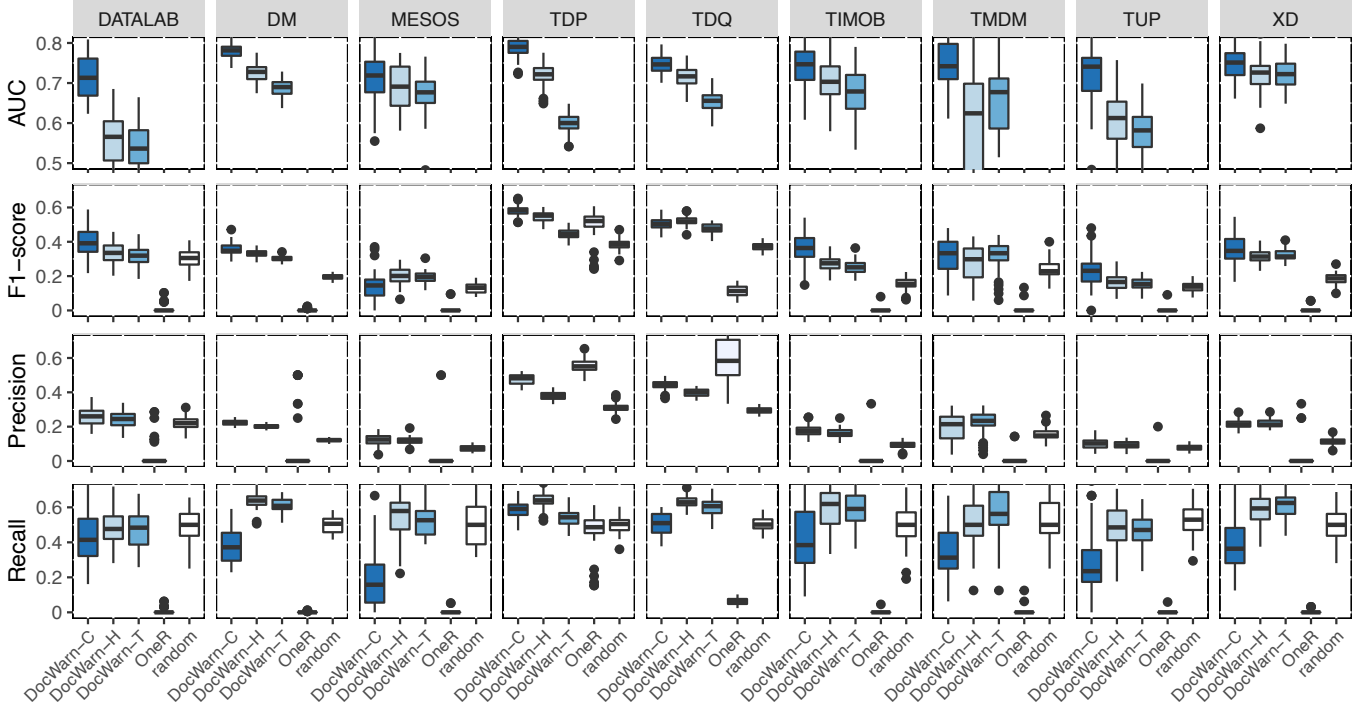


**Figure 4: Distributions of the performance of DocWARN in terms of AUC, F1-score, precision, and recall (50 models built based on 50 cross-validation datasets).**

*permutationImportance* function of the R *mmpf* package [23]. We estimate the MDA for all 50 cross-validation models for each project.

**Ranking the Characteristics:** For each project, we ranked the influence of the metrics based on normalized MDA value ( $\frac{MDA_{metric}}{\sum(MDA_{all})}$ ). Then, we performed the Scott-Knott ESD test to find a statistically distinct rank of each metric across the 50 models. Scott-Knott ESD is a statistical test that partitions a set of means importance values of the metrics into statistically distinct ranks while considering the magnitude of the difference [52]. We used the *sk\_esd* function of the R *ScotKnottESD* package [52].

**Examining the Direction of the Relationship:** We examined the direction of the relationship between an influential metric and the probability that a work item will have a documentation change. To do so, we used DocWARN-C to estimate the probability that a work item will have a documentation change. Then, we used Spearman rank correlation ( $\rho$ ) to measure the relationship of each metric and the probability. A positive correlation indicates a positive relationship between the metric and the probability, while a negative correlation indicates an inverse relationship. We measured the Spearman rank correlation using the *cor.test* function from the R *stats* package [41].

**Result:** The past tendency of developers was highly influential in predicting the documentation changes in DocWARN-C. Figure 5 shows that the stable rates in the past tendency dimension (i.e., *reporter-stable-docchg-rate* and *assignee-stable-docchg-rate*) were ranked as the top three most influential metrics in six studied projects. The Spearman rank correlations test shows an inverse relationship between these metrics and the probability that

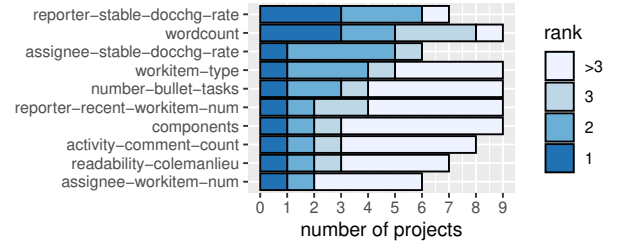


**Figure 5: The ranks of the 10 most influential characteristics (metrics) in DocWARN-C.**

a work item will have a documentation change, i.e., -0.16(TUP) to -0.58(TMDM) for *reporter-stable-docchg-rate* and -0.05(TDP) to -0.6(DM) for *assignee-stable-docchg-rate*. These results indicate that the lower the stable rate, the more likely the work item will have a documentation change. In other words, a work item tends to be unstable if it was reported-by or assigned-to the developers who had worked on the unstable work items. Therefore, the team may include a developer that is familiar with the context of the work item in the sprint planning.

The length of the description text is influential in predicting the documentation changes. Figure 5 shows that *word count* was ranked as the top three most influential metrics in eight projects. The Spearman rank correlations test shows an inverse relationship between word count and the probability that a work item will have a documentation change, i.e., 0.03(DATALAB) to -0.35(TMDM). These

results indicate that the shorter the description text, the more likely a work item will have a documentation change. In other words, a work item with a shorter description text tends to be unstable. Therefore, the team should revisit the work items with a short description text to ensure that they contain sufficient information to understand the work to be done.

**Findings:** The past tendency of developers and the number of words in the description text at the sprint assignment time were the most influential metrics of DocWARN to predict future documentation changes.

**Implications:** The work items with these influential characteristics may need attention before or during the sprint planning as they may have documentation changes in the future.

### 5.3 (RQ3) What are the reasons for the documentation changes in the work items that we can correctly predict?

**Approach:** To qualitatively assess our DocWARN, we manually examine the reasons for the documentation changes in a work item that DocWARN can correctly predict. Since DocWARN-C achieved the best performance, we only focus on the predictions of DocWARN-C. To obtain the list of reasons, we performed open coding on the samples from the three largest projects, i.e., DM, TDQ, and TUP. Then, we manually classified the work items for all studied projects based on the list that we obtained. Below, we describe the open coding and manual classification processes in detail.

**Open Coding:** We used a similar approach as the prior work [11, 17, 33, 43, 60] to perform open coding. In a co-located session, the first and the second authors independently coded the reasons for the documentation changes based on the two versions of the description text. We performed open coding on a batch of 50 work items that are randomly sampled from the pool of work items that will have a documentation change and DocWARN-C can predict correctly. Then, the two authors discussed until the coded reasons were agreed upon. We iteratively performed open coding on a new batch until the list of reasons was saturated, i.e., no new reasons were found in an entire batch. We reached saturation after 150 work items were coded (i.e., three iterations). Lastly, we revisited the 150 work items for another two passes since the late discovered codes might be applied to the earlier work items.

**Manual Classification:** For all nine studied projects, we classified the reasons for documentation changes based on the list of reasons obtained from open coding. To do so, we randomly sampled a representative set of work items that will have a documentation change and that DocWARN-C can predict correctly with a 95% confidence level and 10 confidence interval. Then, the first author classified the reasons for documentation changes using the list of reasons. In total, 394 work items from nine studied projects were classified. Noted that we did not find a new reason for documentation changes after performing manual classification of the nine studied projects.

**Result:** We found that the documentation changes that DocWARN-C can predict were related to four main reasons, i.e., Changing Scope, Defining Scope, Adding Implementation Detail, and Adding Additional Detail. Figure 6 shows the percentage of the five reasons

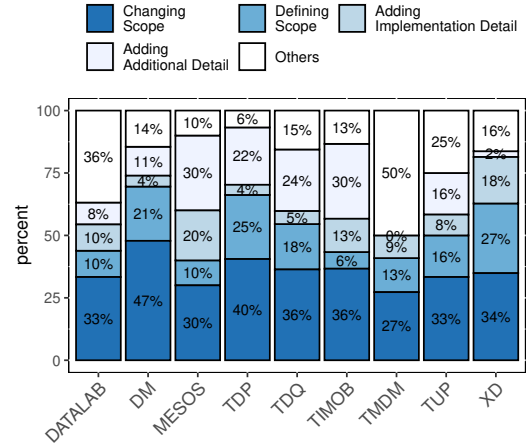


Figure 6: The reasons for the documentation changes that DocWARN-C can predict correctly.

for documentation changes that DocWARN-C can predict correctly. Below, we describe the five reasons for documentation changes.

**Changing Scope:** Figure 6 shows that changing scope was the reason for the documentation changes in 27%(TMDM)-47%(DM) of the sampled work items that were correctly predicted. Scope changing refers to the documentation changes for expanding, reducing, or completely changing the scope of work after work items were assigned to a sprint. For example, DM-4958<sup>7</sup> aimed to implement a function for “region saving”. After the work item was assigned to a sprint, the documented information was changed since its scope of work had been expanded to cover the additional work on the server-side. Prior studies discussed that the change of scope of work could cause the estimated effort and the sprint plan to become inaccurate [35, 38]. We also observed that in DM-4958, i.e., the estimated effort was increased from 6 to 10 or 67%.

**Defining Scope:** Figure 6 shows that defining scope was the reason for the documentation changes in 6%(TIMOB)-27%(XD) of the sampled work items that were correctly predicted. Defining scope refers to the documentation change when the scope of work was not defined (or not clearly defined) at the sprint assignment time. For example, the description of TMDM-10129<sup>8</sup> only described “make it possible to run commandline case on CI”, which was unclear which “commandline” should they focus on. Then, the description was changed to define that the deliverable must be “a test project that can run command line on local server”. A prior study reported that undefined (or delayed) scope of work could affect the team’s ability to estimate effort [19].

**Adding Implementation Detail:** Figure 6 shows that adding implementation detail was the reason for the documentation changes in 4%(DM and TDP)-20%(MESOS) of the sampled work items that were correctly predicted. Adding implementation details refers to the documentation changes for including additional details related to software implementation, which occurred after the work items

<sup>7</sup><https://jira.lsstcorp.org/browse/DM-4958>

<sup>8</sup><https://jira.talendforge.org/browse/TMDM-10129>

were assigned to a sprint. For example, the description of TDQ-16928<sup>9</sup> was only described the scope of work. Then, it was changed to add the implementation details, i.e., a user story, test cases, and implementation steps. However, the scope of work was not modified. Prior work also recommended to maintain a sufficient level of detail in the document to achieve a reliable effort estimation [33, 38].

**Adding Additional Detail:** Figure 6 shows that adding additional detail was the reason for the documentation changes in 0%(TMDM)-30%(MESOS and TIMOB) of the sampled work items that were correctly predicted. Adding additional detail refers to the documentation changes for elaborating on the work to be done. For example, the description text of DM-5794<sup>10</sup> was modified to further explain the use cases of the function to be developed. However, the scope of work and the implementation details were not modified.

**Others:** In addition to the four main reasons above, we found that some documentation changes are trivial or unrelated. For example, the description was changed to rephrase the existing information, log the finished work, format the text, and fix the typo. Figure 6 shows that the documentation changes in 6%(TDP)-50%(TMDM) of the sampled work items that were correctly predicted were related to these reasons.

**Findings:** The reasons for the documentation changes that DocWARN-C can predict were related to changing scope (27%-47%), defining scope (6%-27%), adding implementation detail (4%-20%), and adding additional detail (0%-30%).

**Implications:** DocWARN can identify documentation changes that are related to scope modification, i.e., changing scope and defining scope.

## 6 RELATED WORK

This section discusses the related work in the scope of software effort estimation in Agile, documentation, and automated approaches for effort estimation and sprint planning.

### 6.1 Documentation in Agile Effort Estimation

An agile team used human subjective methods, e.g., Planning Poker, to estimate the effort of a work item [56]. Hence, the team needs to analyze the documented information to understand the work to be done for the work item. Prior studies reported that the quality of documented information could impact the effort estimation accuracy [24, 51, 56, 57]. Furthermore, the documented information may also be changed after the sprint plan is finalized, requiring the team to spend more effort in re-estimating and re-planning the sprint [19, 33, 38].

On the other hand, several studies suggested that the documentation should be provided just-in-time. Ernst and Murphy [12] suggested that the documented information could be elaborated in detail just before or during development. Hence, the documented information may be inadequate for effort estimation [38]. To avoid making unnecessary assumptions, Madampe et al. [33] suggested that the team should spend more effort on detail analysis and documentation. Pasuksmit et al. [38] also suggested that at least the useful information for effort estimation should be documented prior

to finalizing the effort estimates. Meanwhile, Usman et al. [58] proposed a manual checklist to help the team recall relevant factors to improve their understanding of the estimating work items. Motivated by these suggestions of prior work, we developed DocWARN with the goal of helping the Agile team to timely manage uncertainty in documented information, which could impact effort estimation and sprint planning.

### 6.2 Automated Approaches for Effort Estimation and Sprint Planning

Prior work proposed several automated approaches to help the team manage uncertainty in sprint planning. Basri et al. [2] and Shah et al. [50] proposed the approaches to predict the effort to address requirement changes. These approaches focus on formal changes, i.e., the changes submitted via change requests and passed the change impact analysis. Kula et al. [29] used a regression technique to discover the factors that affect the delivery time of *epic*. Dehghan et al. [10] and Choetkiertikul et al. [4] proposed machine learning (ML)-based approaches to predict the delay of work items based on the developers' related metrics and primitive attributes. These studies rely on the metrics extracted from the work items. Unlike these prior studies, this work aims to help an agile team be aware of the potential of documentation changes in a work item, which will enable the team to better manage uncertainty in effort estimation and sprint planning. Moreover, our work is the first to propose the AI/ML-based approaches that use 41 characteristics of work items and the description text to predict the documentation changes.

To help the team estimate effort, prior work proposed automated approaches to predict the effort of a software project [28, 34, 46], an iteration (sprint) [1, 5], and a work item [6, 40, 47]. As these approaches predict the effort based on the documented information, the documentation changes after the work items were assigned to a sprint may lead the predicted effort and the sprint plan to become inaccurate [33, 38]. Our DocWARN will augment these prior effort estimation approaches by determining whether the documented information is stable or not before performing effort estimation.

## 7 THREATS TO THE VALIDITY

This section discusses the threats to the validity of our study.

### 7.1 Construct Validity

In this work, we used an arbitrary threshold to determine the documentation change. Specifically, we determine if a work item had a documentation change if the semantic difference between the last version of description text and its version at sprint assignment time is larger than 10%. Based on our experiments with various thresholds (i.e., 1%-20%), we observed that the lower thresholds tend to include more trivial documentation changes, while few work items had documentation changes larger than 10%. Moreover, our qualitative analysis also showed that only 18% of the correctly predicted documentation changes were trivial (e.g., typo fixing, rephrasing).

When determining the documentation change, we measured the semantic difference of the two versions of description text based on cosine similarity. We opted to use cosine similarity because it computes the similarity in a multi-dimensional space (i.e., a set of words and their frequencies [20]). Nevertheless, using other

<sup>9</sup><https://jira.talendforge.org/browse/TDQ-16928>

<sup>10</sup><https://jira.lsstcorp.org/browse/DM-5794>

similarity metrics (e.g., Jaccard similarity) or other methods (e.g., manual classification) to determine the documentation changes may provide a different set of work items with documentation changes.

We extracted the documentation changes based on the information documented in the JIRA work items. However, it might be possible that the information was changed but the JIRA work item was not updated. To mitigate this threat, we selected to study only the projects that actively maintain the documentation (see Section 4.1). Moreover, the practitioners also reported that they frequently updated the documentation when the new information became available [38].

DocWARN-C used the Random Forest (RF) classification techniques. We opted to use RF instead of deep learning (DL) technique because the RF models can provide a set of important metrics which were used for RQ2, while the DL models cannot. In addition, as shown in Figure 4, DocWarn-H (i.e., using both characteristics and text with the DL model) still achieved a lower performance than DocWarn-C, suggesting that using DL for DocWARN-C is likely to achieve a lower performance as well. Nevertheless, other machine learning techniques may lead to different prediction results. To check this, we built DocWARN-C using another widely-used classification technique, i.e., Logistic Regression. We found that Logistic Regression achieves an average AUC of 0.71 and an average F1-score of 0.37, which is 5% lower and 2% higher than the RF technique, respectively.

## 7.2 Internal Validity

The list of reasons for documentation changes in RQ3 was reported based on the open coding and manual classification performed by the first and the second authors. The personal expertise of the coders might influence the study results. To mitigate this threat, we validated the list of reasons with an external coder (i.e., a software engineer with six years of experience). To do so, the external coder independently categorized a set of 50 randomly selected work items using our list of reasons. Then, we measured the inter-rater reliability using Cohen’s kappa coefficient [37]. The three coders achieved the Cohen’s Kappa value of 0.78 (a substantial agreement), indicating that our results are not subjective to the authors.

To build DocWARN-C and DocWARN-H, we use an automated approach (i.e., AutoSpearman [21]) to remove highly correlated metrics. Manually removing highly-correlated metrics by domain experts may produce a different set of metrics. Nevertheless, we opted to use AutoSpearman to avoid a subjective bias in the metrics removal process and to produce a consistent set of metrics for the sake of reproducibility.

We analyzed the influential characteristics of a work item by analyzing the Random Forest classifiers of DocWARN-C and their estimated probabilities. The observed relationship between the influential characteristics of a work item and the probability that a work item will have documentation changes does not represent a causal effect. The real causes of documentation changes are challenging to determine solely using a prediction technique. Therefore, future studies are needed to verify the causal effect of our findings.

We validate the performance of DocWARN using a cross-validation approach. This approach may cause a data leak, i.e., the newer work items may become training data to predict the older work items.

Nevertheless, all the characteristics of a work item used in this paper were extracted using only the prior work items (i.e., the ones created before the work item under the study). Hence, all the predictions of DocWarn-C are based on past data (not the future).

## 7.3 External Validity

In this study, we proposed and evaluated DocWARN based on the nine open-source software projects. These studied projects are developing the software in sprints and actively documenting their work items in an online task management system. The demographics of these projects are also diverse in terms of size, context, development teams, and the community. Nevertheless, our DocWARN may achieve different performances in projects with different settings, e.g., the projects that do not work in sprints, use other approaches for documentation, do not perform effort estimation, or do not record the change history of work items. Hence, further study is needed to confirm our findings for those projects.

## 8 CONCLUSION

Effort estimation and sprint planning are performed based on the available documented information [38, 48]. Thus, it is possible that documentation changes could invalidate the original understanding, which in turn will negatively impact the effort estimation accuracy and sprint plan. Hence, to help an agile team to better manage the uncertainty and reduce the risk of unreliable estimated effort and sprint plan, we proposed an approach called “**DocWARN**” to predict documentation changes of a work item after it was assigned to a sprint. DocWARN achieved an average AUC of 0.75 and an average F1-Score of 0.36, which is significantly better than the baselines. We found that the past tendency of developers and the length of description text are the most influential characteristics of a work item for determining documentation changes. We also found that 40%–68% of the documentation changes that DocWARN can correctly predict were related to scope modification. To help practitioners better manage the uncertainty, future work should further explore approaches that can predict the documentation changes for the finer-grained document (e.g., a specific part of documented information) and can estimate the potential impact of documentation changes. Such approaches should help practitioners focus on the important changes that lead to a significant impact on the estimated effort and sprint plan.

**Acknowledgement.** P. Thongtanunam was partially supported by the Australian Research Council’s Discovery Early Career Researcher Award (DECRA) funding scheme (DE210101091).

## REFERENCES

- [1] Pekka Abrahamsson, Raimund Moser, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. 2007. Effort Prediction in Iterative Software Development Processes – Incremental Versus Global Prediction Models. In *ESEM*. 344–353.
- [2] Sufyan Basri, Nazri Kama, Faizura Haneem, and Saiful Adli Ismail. 2016. Predicting effort for requirement changes during software development. In *Proc. of the SoICT*. 380–387.
- [3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
- [4] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. 2017. Predicting the delay of issues with due dates in software projects. *EMSE* 22, 3 (2017), 1223–1263.

- [5] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Aditya Ghose, and John Grundy. 2017. Predicting delivery capability in iterative software development. *TSE* 44, 6 (2017), 551–573.
- [6] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Thi Minh Pham, Aditya Ghose, and Tim Menzies. 2019. A deep learning model for estimating story points. *TSE* 45, 07 (2019), 637–656.
- [7] Evita Coelho and Anirban Basu. 2012. Effort Estimation in Agile Software Development using Story Points. *IJAIS* 3, 7 (2012), 7–10.
- [8] Mike Cohn. 2006. *Agile estimating and planning*.
- [9] Meri Coleman and T L Liau. 1975. A Computer Readability Formula Designed for Machine Scoring. *Journal of Applied Psychology* 60, 2 (1975), 283–284.
- [10] Ali Dehghan, Adam Neal, Kelly Blincoe, Johan Linaker, and Daniela Damian. 2017. Predicting Likelihood of Requirement Implementation within the Planned Iteration: An Empirical Study at IBM. In *Proc. of the MSR*. 124–134.
- [11] Farida El Zanaty, Toshiaki Hirao, Shane McIntosh, Akinori Ihara, and Kenichi Matsumoto. 2018. An Empirical Study of Design Discussions in Code Review. In *Proc. of the ESEM*. 11:1–11:10.
- [12] Neil A Ernst and Gail C Murphy. 2012. Case Studies in Just-In-Time Requirements Analysis. In *Proc. of the EmpiRE*. 25–32.
- [13] Rudolph Flesch. 1948. A New Readability Yardstick. *Journal of applied psychology* 32, 3 (1948), 221–233.
- [14] Martin Fowler and Jim Highsmith. 2001. The agile manifesto. *Software Development* 9, 8 (2001), 28–32.
- [15] Aaron Gokaslan and Vanya Cohen. 2019. OpenWebText Corpus. <http://Skylion007.github.io/OpenWebTextCorpus>
- [16] R Gunning. 1952. *The Technique of Clear Writing*. McGraw-Hill, New York.
- [17] Marlo Haering, Christoph Stanik, and Walid Maalej. 2021. Automatically Matching Bug Reports With Related App Reviews. In *Proc. of the ICSE*. 970–981.
- [18] Jun Han and Claudio Moraga. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks*. 195–201.
- [19] Rashina Hoda and Latha K. Murugesan. 2016. Multi-Level Agile Project Management Challenges: A Self-Organizing Team Perspective. *JSS* 117 (2016), 245–257.
- [20] Anna Huang et al. 2008. Similarity measures for text document clustering. In *Proc. of the NZCSRSC*, Vol. 4. 9–56.
- [21] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude. 2018. AutoSpearman: Automatically Mitigating Correlated Software Metrics for Interpreting Defect Models. In *Proc. of the ICSME*. 92–103.
- [22] Jonathan Anderson. 1983. Lix and Rix: Variations on a Little-known Readability Index. *Journal of Reading* 26, 6 (1983), 490–496.
- [23] Zachary M. Jones. 2018. mmpf: Monte-Carlo Methods for Prediction Functions. *The R Journal* 10, 1 (2018), 56–60.
- [24] Magne Jorgensen, Barry Boehm, and Stan Rifkin. 2009. Software Development Effort Estimation: Formal Models or Expert Judgment? *IEEE Software* 26, 2 (2009), 14.
- [25] Rashidah Kasauli, Grisha Liebel, Eric Knauss, Swathi Gopakumar, and Benjamin Kanagwa. 2017. Requirements Engineering Challenges in Large-Scale Agile System Development. In *RE*. 352–361.
- [26] J Peter Kincaid, Robert P Fishburne Jr, Richard L Rogers, and Brad S Chissom. 1975. *Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel*.
- [27] Eriks Klotins, Michael Unterkalmsteiner, Panagiota Chatzipetrou, Tony Gorschek, Rafael Prikladnicki, Nirnaya Tripathi, and Leandro Pompermaier. 2021. A Progression Model of Software Engineering Goals, Challenges, and Practices in Start-Ups. *TSE* 47, 3 (2021), 498–521.
- [28] S Kuan. 2017. Factors on software effort estimation. *IJSEA* 8, 1 (2017), 23–32.
- [29] Elvan Kula, Arie van Deursen, and Georgios Gousios. 2021. Modeling Team Dynamics for the Characterization and Prediction of Delays in User Stories. In *Proc. of the ASE*.
- [30] A. Liaw and M. Wiener. 2018. *Package 'randomForest': Breiman and Cutler's random forests for classification and regression*. <https://CRAN.R-project.org/package=randomForest> R package version 4.6-14.
- [31] Andy Liaw, Matthew Wiener, et al. 2002. Classification and Regression by randomForest. *R news* 2, 3 (2002), 18–22.
- [32] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692* (2019).
- [33] Kashumi Madampe, Rashina Hoda, and John Grundy. 2020. A Multi-dimensional Study of Requirements Changes in Agile Software Development Projects. *arXiv preprint arXiv:2012.03423* (2020).
- [34] I Manga and NV Blamah. 2014. A particle Swarm Optimization-based Framework for Agile Software Effort Estimation. *IJES* 3, 6 (2014), 30–36.
- [35] Zainab Masood, Rashina Hoda, and Kelly Blincoe. 2020. Real World Scrum, A Grounded Theory of Variations in Practice. *TSE* (2020), To appear.
- [36] G Harry Mc Laughlin. 1969. SMOG Grading-a New Readability Formula. *Journal of Reading* 12, 8 (1969), 639–646.
- [37] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.
- [38] Jirat Pasuksmit, Patanamon Thongtanunam, and Shanika Karunasekera. 2021. Towards Just-Enough Documentation for Agile Effort Estimation: What Information Should Be Documented? *Proc. of the ICSME* (2021), 114–125.
- [39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
- [40] Simone Porru, Alessandro Murgia, Serge Demeyer, Michele Marchesi, and Roberto Tonelli. 2016. Estimating Story Points from Issue Reports. In *Proc. of the PROMISE*. 2:1–2:10.
- [41] R Core Team. 2013. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing.
- [42] Payam Refaellizadeh, Lei Tang, and Huan Liu. 2009. Cross-validation. *Encyclopedia of database systems* 5 (2009), 532–538.
- [43] Peter C. Rigby and Margaret Anne Storey. 2011. Understanding Broadcast Based Peer Review on Open Source Software Projects. In *Proc. of the ICSE*. 541–550.
- [44] Kenneth S Rubin. 2012. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*.
- [45] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv abs/1910.01108* (2019).
- [46] Federica Sarro and Alessio Petrozziello. 2018. Linear Programming as a Baseline for Software Effort Estimation. *TOSEM* 27, 3 (2018), 12:1–12:28.
- [47] Ezequiel Scott and Dietmar Pfahl. 2018. Using Developers' Features to Estimate Story Points. In *Proc. of the ICSSP*. 106–110.
- [48] Todd Sedano, Paul Ralph, and Cécile Péraire. 2019. The Product Backlog. In *Proc. of the ICSE*. 200–211.
- [49] R. Senter and E.A.Smith. 1967. *Automated Readability Index*. Technical Report. University of Cincinnati.
- [50] Jalal Shah, Nazri Kama, and Nur Azaliah A Bakar. 2018. A Novel Effort Estimation Model For Software Requirement Changes During Software Development Phase. *IJSEA* 9, 6 (2018).
- [51] Richard Bernrtsson Svensson, Tony Gorschek, Per-Olof Bengtsson, and Jonas Widerberg. 2019. BAM-Backlog Assessment Method. In *Proc. of the XP*. 53–68.
- [52] Chakkrit Tantithamthavorn. 2018. *ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test*. <https://CRAN.R-project.org/package=ScottKnottESD> R package version 2.0.3.
- [53] Chakkrit Tantithamthavorn and Ahmed E Hassan. 2018. An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges. In *Proc. of the ICSE-SEIP*. 286–295.
- [54] Chakkrit Tantithamthavorn, Ahmed E Hassan, and Kenichi Matsumoto. 2018. The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models. *TSE* (2018), 99.
- [55] Maciej Tomczak and EWA Tomczak. 2014. The need to report effect size estimates revisited. An overview of some recommended measures of effect size. *Trends in Sport Sciences* 21, 1 (2014), 19–25.
- [56] M. Usman and R. Britto. 2016. Effort Estimation in Co-located and Globally Distributed Agile Software Development: A Comparative Study. In *Proc. of the IWSM-MENSURA*. 219–224.
- [57] Muhammad Usman, Ricardo Britto, Lars-Ola Damm, and Jürgen Börstler. 2018. Effort Estimation in Large-Scale Software Development: An Industrial Case Study. *IST* 99 (2018), 21–40.
- [58] Muhammad Usman, Kai Petersen, Jürgen Börstler, and Pedro Santos Neto. 2018. Developing and using checklists to improve software effort estimation: A multi-case study. *JSS* 146 (2018), 286–309.
- [59] Holger von Joanne-Diedrich. 2017. OneR: One rule machine learning classification algorithm with enhancements. *R package version 2* (2017), 2.

- [60] Supatsara Wattanakriengkrai, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Hideaki Hata, and Kenichi Matsumoto. 2020. Predicting Defective Lines Using a Model-Agnostic Technique. *TSE (2020)*, To appear.
- [61] Frank Wilcoxon. 1992. Individual Comparisons by Ranking Methods. In *Break-throughs in statistics*. 196–202.
- [62] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proc. of the EMNLP*. 38–45.