mathematics

MDPI

Article

Developer Assignment Method for Software Defects Based on Related Issue Prediction

Baochuan Liu, Li Zhang, Zhenwei Liu and Jing Jiang *

State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China; liubc@buaa.edu.cn (B.L.); lily@buaa.edu.cn (L.Z.); meetliuzhenwei@gmail.com (Z.L.)
* Correspondence: jiangjing@buaa.edu.cn

Abstract: The open-source software platform hosts a large number of software defects, and the task of relying on administrators to manually assign developers is often time consuming. Thus, it is crucial to determine how to assign software defects to appropriate developers. This paper presents DARIP, a method for assigning developers to address software defects. First, the correlation between software defects and issues is considered, predicting related issues for each defect and comprehensively calculating the textual characteristics of the defect using the BERT model. Second, a heterogeneous collaborative network is constructed based on the three development behaviors of developers: reporting, commenting, and fixing. The meta-paths are defined based on the four collaborative relationships between developers: report–comment, report–fix, comment–comment, and comment–fix. The graph-embedding algorithm metapath2vec extracts developer characteristics from the heterogeneous collaborative network. Then, a classifier based on a deep learning model calculates the probability assigned to each developer category. Finally, the assignment list is obtained according to the probability ranking. Experiments on a dataset of 20,280 defects from 9 popular projects show that the DARIP method improves the average of the Recall@5, the Recall@10, and the MRR by 31.13%, 21.40%, and 25.45%, respectively, compared to the state-of-the-art method.

Keywords: defect fixing; developer assignment; prediction of related issues; heterogeneous collaborative network

MSC: 68M20

Citation: Liu, B.; Zhang, L.; Liu, Z.; Jiang, J. Developer Assignment Method for Software Defects Based on Related Issue Prediction. *Mathematics* **2024**, *12*, 425. https://doi.org/10.3390/math12030425

Academic Editors: Mingbo Zhao, Haijun Zhang, Zhou Wu and Faheim Sufi

Received: 15 December 2023
Revised: 24 January 2024
Accepted: 26 January 2024
Published: 28 January 2024

1. Introduction

GitHub is a web-based project-hosting platform [1–3] that provides an Issue-Tracking System to help users manage issues that occur in their projects. In the Issue-Tracking System [4,5], users can report software defects found during development [6–8], project managers can assign software defects to appropriate developers, and other developers can participate in discussions around defects of interest. However, with over 200 million open-source projects and 1.2 billion issues hosted on the GitHub platform worldwide, there are a large number of software defects that need to be fixed every day. Manually assigning developers to software defects by managers alone can be a tedious and time-consuming task [9–12]. To streamline this process, researchers have conducted studies on the task of assigning or recommending developers [13–22]. For example, Yang et al. [13] filtered developers from historical defects by calculating textual similarities between different software defects, and then assigned developers to software defects based on their development experience. Aung et al. [14] processed code snippets and text information in defect reports separately and improved the performance of developer assignments through joint learning of developer assignment tasks and label classification tasks.

Empirical studies have shown that some software defects may be related [23,24], and two related software defects could involve the same source code file. Almhana and Kessentini [25] define dependencies based on two software defects involving the same source code

file. They leverage the dependencies to help resolve the task of developer assignment on the Mozilla platform. Jahanshahi et al. [26] discovered that some software defects may be blocked by others during the repair process. They studied how to assign developers for software defects based on their blocking relationships. Since there are correlations between software defects for reasons such as involving the same source code files or blocking relationships, referring to related software defects may help assign appropriate developers to software defects. However, most existing studies only consider the characteristics of the current software defect itself and overlook the correlation between software defects when automatically assigning developers to software defects.

Meanwhile, existing studies do not consider developers' different development behaviors when collecting candidates and extracting their characteristics [27,28]. For example, Xuan et al. [28] considered the commenting behavior among developers to build social networks and designed a model for assigning developer priorities. Considering the different behaviors of developers can help to understand their characteristics, which can help to assign the appropriate developer to a software defect.

This paper proposes an automated Developer Assignment method for open-source software defects based on Related Issue Prediction, named DARIP. First, this paper extracts the text vectors of software defects and all their historical software defects using the BERT model, predicts the Top-N potential related software defects for each software defect from all the historical software defects based on cosine similarity, takes the text information of related software defects as an extension of the text information of software defects, and takes the text information of software defects together with the text information of related software defects as text input. The text characteristics of software defects are obtained by comprehensively calculating the text vectors of software defects and related software defects. Second, a heterogeneous collaborative network is constructed based on the three development behaviors of developers: reporting, commenting, and fixing. The meta-paths are defined based on the four collaborative relationships between developers: report–comment, report–fix, comment–comment, and comment–fix. The graph-embedding algorithm metapath2vec extracts developer characteristics from the heterogeneous collaborative network. Then, a classifier based on a deep learning model calculates the probability assigned to each developer category. Finally, the assignment list is obtained according to the probability ranking.

In order to evaluate the effectiveness of the DARIP method, this paper collects 20,280 software defects from 9 popular open-source projects. The experimental results show that the average values of the Recall@5, the Recall@10, and the MRR of the method DARIP in 9 projects are 0.5902, 0.7160, and 0.4136, respectively. Compared with the existing method Multi-triage, the average values of the Recall@5, the Recall@10, and the MRR of the DARIP method in 9 projects are increased by 31.13%, 21.40%, and 25.45%, respectively.

2. Background

In the GitHub open-source community, the Issue Tracker System supports developers to contribute freely to open-source projects. In the Issue Tracker System, users can report software defects or functional requirements found during the development process. For example, managers may identify an issue as a software defect [29] by adding the label "bug", "defect" or other synonyms. After the issue is reported, other interested developers participate in the issue comments to discuss the solution. In response to the reported software defects, managers have the option of assigning appropriate developers. Moreover, developers can create a pull request to submit fixable code for code review based on the solution to a software defect. After passing the code review, the code is merged into the repository, and the software defect is fixed. Therefore, this paper refers to the literature [14], which defines the fixer of a software defect as the assignee or the creator who creates the corresponding pull request and is successfully merged into the repository.

In this paper, we study the software defects in the Issue-Tracking System. Take the software defect of No. 29391 (https://github.com/tensorflow/tensorflow/issues/29391,

(accessed on 14 December 2023)) in the project *tensorflow/tensorflow* as an example, and its basic structure is shown in Figure 1.

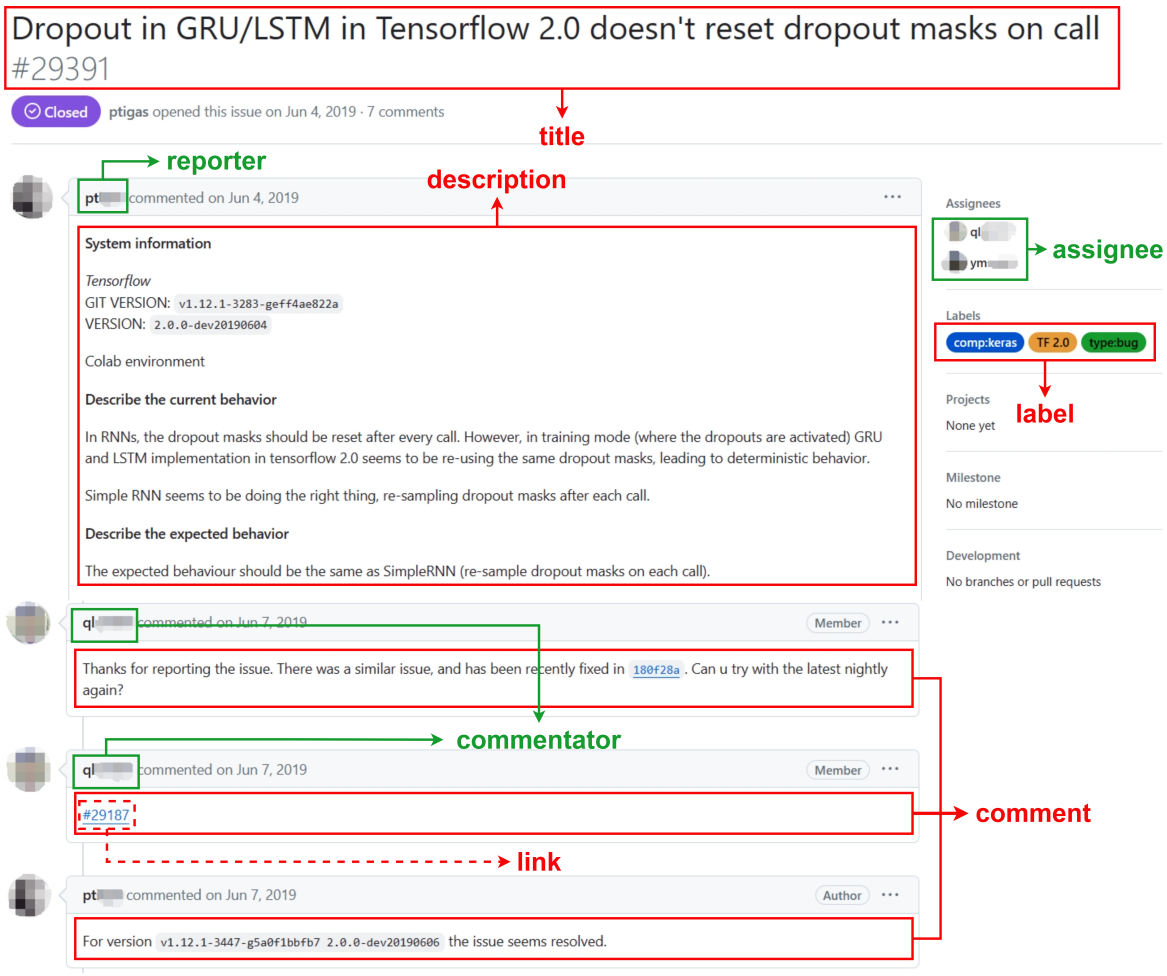

Figure 1. A sample of a software defect.

(1) Title: A brief overview of the software defect.

(2) Description: The body of the defect report, including a specific description of the software defect.

(3) Comment: Information about developers participating in discussions about a software defect.

(4) Label: Managers add labels to identify the category of a issue. For example, add the label *type:bug* to indicate that the current issue is a software defect.

(5) Reporter: The developer who reported the current software defect.

(6) Commentator: Developers involved in the discussion of a software defect.

(7) Assignee: A developer assigned by the manager to fix the software defect.

(8) Link: A hyperlink left by a developer during the discussion of an issue, often used to refer to other relevant information.

As shown in Figure 1, the developer *pt***** reported a software defect number 29391, which was assigned to developers *ql****** and *ym***** on the project *tensorflow/tensorflow*. Subsequently, the developer *ql****** indicated that the current software defect was related to the historical software defect number 29187 (https://github.com/tensorflow/tensorflow/issues/29187, (accessed on 14 December 2023)) in the same project, and the submitted code had fixed the related software defect. By validation from the developer *pt*****, the current software defect can also be fixed using the same version of the fixed code. The historically related software defect number 29187 was reported earlier than the software defect number

29391 and has been fixed by the developer *ql******. In particular, the assignee *ql****** of the software defect number 29391 was also the assignee of the related software defect number 29187. Already familiar with the resolution of the historically related software defect number 29187, the developer *ql****** immediately fixed the software defect number 29391 with the same fixable code.

As a related issue, the resolution of software defect number 29391 helped to fix the software defect number 29187. At the same time, the experience of developer *ql****** in fixing related issues helped reduce the resolution time for software defect number 29187. Therefore, referring to the issues associated with the software defect may help to assign the appropriate developer to the software defect.

3. Related Work

3.1. Related Issue

Due to the fact that open-source software can share or reuse code, many open-source projects rely on the same libraries and components [12]. Dependencies between open-source projects further affect software defects in the project, and issues in one project may be linked to related issues from the same or other projects [30–33]. Ma et al. [30] conducted an empirical study of software defects in seven open-source projects in the Python ecosystem on the GitHub platform. They found that developers point to related issues by leaving links in the text messages of software defects. Zhang et al. [31] performed both qualitative and quantitative analysis of links in the Rails ecosystem and found that the percentage of issues with links reached 24.8%. Of these, 82.8% were links within projects, and 17.2% were links between different projects. Li et al. [32] analyzed 16,584 Python projects on the GitHub platform. They classified the relationship between software defects and related issues into six different types through qualitative analysis, and each type of related issue could help to fix software defects. Previous studies have shown that referencing related issues may help fix corresponding software defects.

Zhang et al. [34] stated that issue reports on GitHub are essential knowledge in software development and proposed iLinker, a method to automatically obtain related issues to help fix issues by sharing knowledge of related issues. The Linker method calculated three similarity scores between query issues and candidate issues using TF-IDF, word embedding, and document embedding, respectively. Then, the final scores of the query issues and the candidate issues are combined, and the related issues of query issues are recommended based on the final scores. In our previous work [35], we focused on software defects and related issues and proposed an algorithm, CPIRecom, to predict cross-project related issues automatically. Firstly, the pre-selecting set construction method is proposed to filter the massive issues. Secondly, the BERT pre-training model is used to extract text features and analyze project features. Then, the random forest algorithm is used to calculate the probability of the pre-selecting issues and the software defects. Finally, the recommendation list is obtained according to the ranking.

In this paper, we draw on the idea of the CPIRecom method of related issue recommendation to predict potentially related issues for each software defect in order to help solve the task of developer assignment. Our existing work and our current work are different because they are designed to solve different research questions. Our prior work [35] aimed at predicting relevant issues from other projects for software defects. However, the purpose of the current work is to assign appropriate developers for software defects.

3.2. Developer Assignment

In this section, we present existing research on developer assignment and divide existing work into four categories: information retrieval-based approaches, machine learning-based approaches, social network-based approaches, and defect relationship-based approaches.

3.2.1. Information Retrieval-Based Approach

Information retrieval refers to the process of finding information from a collection of data that meets information needs. The developer assignment method based on information retrieval assumes that developers can fix specific types of software defects because they have some specialized knowledge. Some researchers introduced the concept of topic model for assigning developers [13,15–17]. Among them, Yang et al. [13] proposed a developer assignment method with a multi-feature combination, extracted topics from historical defect reports, extracted features such as components, products, priorities, and severity of software defects, calculated textual similarities among software defects, and collected developers who contributed as candidates. Finally, they obtained a ranking list of candidate developers based on development experience. Xia et al. [16] proposed a multi-featured topic model that extends the Latent Dirichlet Allocation(LDA) to the developer recommendation task. They considered product information and component information of software defects and assigned appropriate developers to software defects based on the affinity of the developer to the topics.

Most of the above information retrieval-based methods are divided into two steps: filtering out a small number of eligible historical software defects and collecting all the developers from them to obtain a candidate set, then extracting the characteristics of all the candidates in the candidate set and finally assigning appropriate developers to the software defects. If the real developers do not appear in the candidate set, it will directly affect the effectiveness of the method.

3.2.2. Machine Learning-Based Approach

With the continuous development of machine learning technology, some researchers utilized traditional machine-learning and deep-learning algorithms to solve the developer assignment task [14,18–22,36–39]. Jonsson et al. [19] proposed a defect assignment method based on ensemble learner, using TF-IDF technology to extract features from the title information and description information of defect reports and combining Naive Bayes, Support Vector Machines, KNN, and Decision Tree classifiers to create a stacked generalization classifier for improving the prediction accuracy of automatic developer assignment. Lee et al. [21] propose a developer assignment method based on a CNN and word embedding, using the word2vec model to convert each word in the summary and description in defect reports into a word vector and then using a CNN to learn the text features of software defects to solve the defect classification task. Mani et al. [22] proposed a developer assignment method based on deep bidirectional recurrent neural networks, which learns paragraph-level representations of software defects based on deep learning algorithms, learned word order and semantic relationships from the context of defect reports, and finally, realized defect classification using a softmax classifier. Aung et al. [14] proposed a multi-task learning classification model using a text encoder to extract text features from defect reports and an AST encoder to extract code features from code snippets to improve the performance of the model by jointly learning defect assignment and label classification tasks.

Most of the existing methods based on machine learning use various models to learn feature representations from the text information of software defects and apply the text classification to solve the developer assignment task. However, many of these methods only focus on the software defects themselves and overlook the correlation between them. Therefore, it is crucial to consider the related issues of software defects when extracting textual characteristics, as this can aid in solving the task of developer assignment by extending the text information of software defects.

3.2.3. Social Network-Based Approach

Some researchers utilized methods based on social network analysis to solve the task of developer assignment [14,40–45]. For example, Banitaan and Alenezi [40] proposed DECOBA, a developer assignment method based on developer communities. They used de-

velopers' commenting behavior on software defects to construct a developer social network. Then, they detected developer communities and ranked developers by their experience in each community to obtain a developer assignment list. Zhang et al. [41] took commentators on software defects as potential developers. They constructed heterogeneous social networks based on developers, software defects, comments, components, and products. They then calculated three types of collaborative proximity among developers in these networks based on defects, components, and products. The researchers ranked developers according to their comprehensive score and generated a recommended list of potential developers. Zaidi and Lee [42] proposed a graphical representation of defect reports, extracted the title and description information from the defect reports, took the defect report as a document, the document and the word in a document as two types of nodes, and built a heterogeneous network with the document-to-word and the word-to-word as two types of edges. The Graph Convolutional Network was used to learn the graph representation of software defects, and the task of defect classification was solved by using node classification.

The social network-based approaches mentioned above did not consider the reporting, commenting, and fixing behavior of developers in software defects. Therefore, it is essential to analyze the personal characteristics of developers when extracting developer characteristics for software defects. This analysis can help in assigning appropriate developers to software defects.

3.2.4. Defect Relationship-Based Approach

A few researchers consider the relationship between software defects in the developer assignment task of studying software defects [12,25,26]. Almhana and Kessentini [25] proposed a defect classification method based on the dependency between defect reports, which defined the dependency between two defect reports as the number of shared files to be inspected to localize the defects. Then, they adopted a multi-objective search to rank the bug reports for developers based on both of their priorities and the dependency between them. Jahanshahi et al. [26] introduced a defect-triaging method called DABT, which considered the text information, the cost associated with each defect, and the dependency among them. They leveraged natural language processing and integer programming to assign software defects to appropriate developers.

Although the defect relationship-based methods mentioned above have proven to be effective, they rely on platforms like Mozilla and Bugzilla that contain information such as dependency relationships and source code files. Unfortunately, the GitHub platform lacks this information, making it impossible to apply these methods directly to it. Therefore, it is necessary to draw on the related issue recommendation method to predict the potential related issues for each software defect to assist in the task of assigning developers.

4. Method

Given the difficulty of manually assigning developers to software defects in GitHub, this paper proposes a method for automatically assigning developers to software defects. Then, we present the design idea and the individual modules of the DARIP method.

4.1. Design Idea

After a software defect is reported, it may be assigned to different developers. This allows us to label the type of software defect as the assignee and convert the developer assignment task into a multi-classification task in the field of machine learning. In this paper, we propose the automated developer assignment method, DARIP, which is divided into two phases: the training phase and the testing phase. The overall architecture is shown in Figure 2. The DARIP methodology consists of the following components:

(1) Collect historical software defects and developers.

(2) Predict potentially related issues for software defects based on their textual information.

(3) The feature vectors of text information in software defects and the feature vectors of text information in related issues are used to calculate the textual features of software defects comprehensively.

(4) Consider the three development behaviors of developers in software defects: reporting, commenting, and fixing. Construct a heterogeneous collaborative network with developers and software defects as two types of nodes and three development behaviors as three types of edges.

(5) A graph-embedding algorithm is used to extract the developer features of software defects from heterogeneous collaborative networks.

(6) The textual features and textual features of software defects are input into the classifier for training and learning, and the developer assignment model is obtained.

(7) Extract the text feature and developer feature for a newly reported software defect.

(8) Input the extracted text feature and developer feature into the trained developer assignment model and output the developer assignment list.

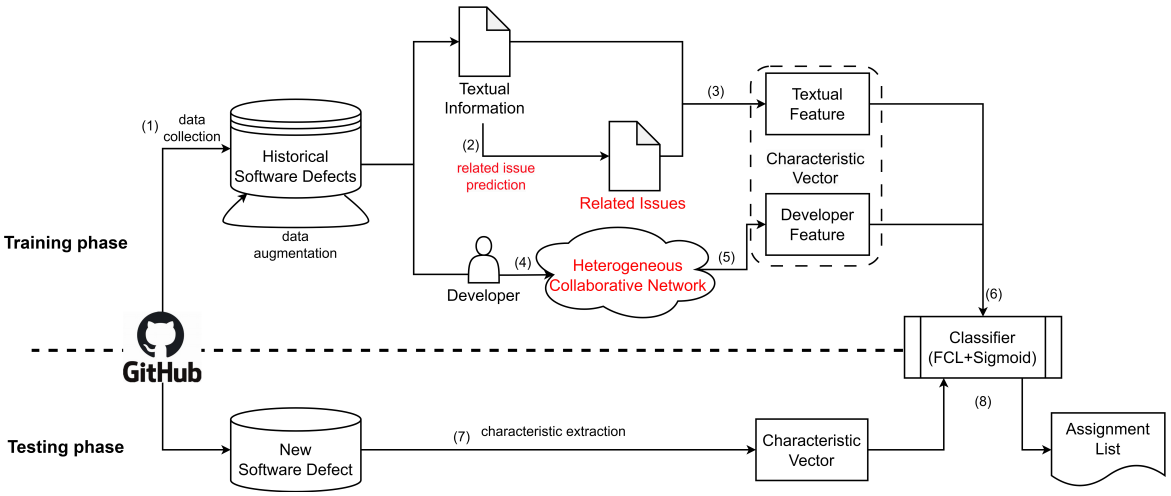

Figure 2. Overall procedure of DARIP method.

In the training phase, this paper first collects software defects from the GitHub platform and the developers. Then, potentially related issues are predicted for software defects, and text features are calculated comprehensively. At the same time, a heterogeneous collaborative network is constructed based on three development behaviors of developers: reporting, commenting, and fixing. The graph-embedding algorithm is used to learn the feature representation of developers. Finally, the feature vectors of software defects are input to the classifier for training and learning, and the developer assignment model is obtained.

During the testing phase, when new software defects are reported on the GitHub platform, the DARIP method extracts the textual and developer characteristics of the software defects. It then calculates the probability of each candidate developer being able to fix the software defects using the assignment model generated during the training phase. Finally, the candidate developers are ranked based on their probability, from highest to lowest, to create an assignment list of potential developers.

4.2. Textual Characteristic

When developers report software defects, they provide detailed information about the problem using both the title and description. Therefore, text information is an essential reference to help assign developers. At the same time, a software defect in open-source software may be related to other issues, and referring to the related issues may help fix the software defect. Therefore, the following section describes how to extract the textual characteristic.

To prepare the text information of software defect for analysis, we first preprocess the text information. The pre-processing includes removing non-text information, converting all letters to lowercase, removing stop words, segmenting the text, and lemmatization. In our study, we remove non-text information such as phone numbers, email addresses, and emoticons from the title and description. We then use blank spaces to separate the remaining text into individual words, creating an array of words for analysis.

Secondly, for a software defect, the potentially related issue is predicted, and the text information of the related issue is taken as an extension of the text information of the software defect. The text information of the software defect and the text information of the related issue are taken as the text input. Specifically, the DARIP method collects all historical software defects corresponding to each software defect from the dataset. Then, natural language processing techniques are used to extract the textual vectors of software defects and all their historical software defects. Currently, the BERT pre-training model [46] is an important technique in the field of natural language processing. We use the BERT pre-training model to transform the preprocessed word arrays into text vectors that can characterize the semantic information. Then, we use cosine similarity to calculate the textual similarity of feature vectors between the software defect and each historical software defect, rank all its historical software defects from highest to lowest according to the textual similarity, and select the Top-N historical software defects with the highest textual similarity as the potential N related issues of the software defect. Extend the text information of related issues as the text information of software defect, and take the text information of software defect and related issues together as text input.

Finally, the textual vectors of software defects and the textual vectors of Top-N related issues are weighted to get the final textual characteristic of the software defect. Specifically, the cosine similarity between the textual vector of the related issues and the textual vector of the software defect is calculated as the correlation coefficient of the related issues. For each related issue, the weighted value is then obtained by multiplying its textual vector with the correlation coefficient. Next, we calculate the cumulative weighted value of N related issues. Then, we calculate the sum of the cumulative weighted values of N related issues and the textual vector of software defect. Finally, divide the sum by N+1 to obtain the final textual feature of the software defect. The formula for calculating the textual characteristic of software defect is as follows:

$$tFeat(i) = \frac{1}{N+1}\left(bf(i) + \sum_{k=1}^{N}(Cosine(i)_k * bf(i)_k)\right) \quad (1)$$

where N represents the number of potentially related issues predicted by the i-th software defect, $bf(i)$ represents the text vector of the i-th software defect, $bf(i)_k$ represents the text vector of the i-th software defect corresponding to the k-th related issue, $Cosine(i)_k$ represents the cosine similarity between the i-th software defect and the corresponding k-th related issue, and $tFeat(i)$ represents the final textual characteristic of the i-th software defect. According to Section 6.3, the value of N is set to 1. This means that the text information of a software defect is used as textual input together with the text information of the TOP-1 related issue.

4.3. Developer Characteristic

The development behaviors of developers in software defects mainly include reporting, commenting, and fixing, and developers can participate in the development of different software defects based on different development behaviors according to their personal wishes. For the newly reported software defects, the DARIP method considers the three development behaviors of developers: reporting, commenting, and fixing. Based on the three development behaviors of developers in a software defect and its historical software defects, a heterogeneous collaborative network is constructed. The graph-embedding algorithm is then used to mine the personal features of developers from the heterogeneous collaborative network and learn the vector representations of developers. Finally, the feature vector of

the reporter of the new software defect is used as the developer characteristic of the current software defect.

4.3.1. Heterogeneous Collaborative Network

First, for a software defect and all corresponding historical software defects, collect the participating developers, including reporters, commentators, and fixers. Then, based on the three development behaviors of reporting, commenting, and fixing software defects, a heterogeneous collaborative network is built corresponding to current software defect, all historical software defects, and all developers involved in the above defects. In this paper, we consider assigning appropriate developers to the software defect as soon as they are reported, so that the only participant in newly reported software defects is the reporter.

The definition of a heterogeneous collaborative network is shown in Table 1. The flowchart for constructing a heterogeneous collaborative network for a software defect is shown in Figure 3. First, we collect all historical software defects. Second, we select a software defect to be addressed from the historical software defects. Third, we collect the developers from the software defect, including reporter, commentator, and fixer. Fourth, we construct the reporting relationship, the commenting relationship, and the fixing relationship between developers and software defects, respectively. Fifth, we determine whether there are any software defects to be addressed. If so, go back to the second step. If not, it indicates that all software defects have been handled and the final heterogeneous collaborative network is output.

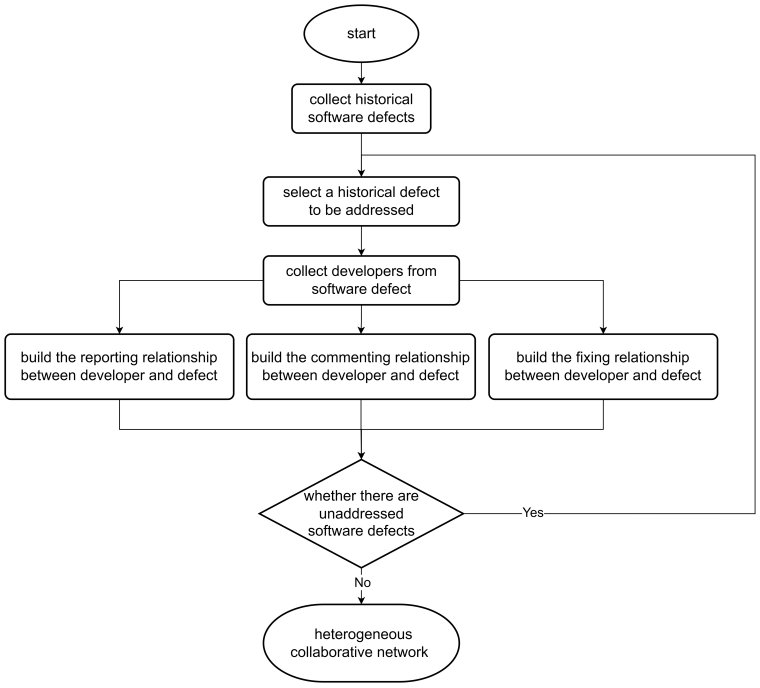

Figure 3. The flowchart for construct heterogeneous collaborative network.

In heterogeneous collaborative networks, the node types include software defects and developers, and the edge types include developers' reporting behavior, commenting behavior, and fixing behavior. We represent a software defect as B (also known as _Bug), a _Developer as D, a _reporting behavior as r, a _commenting behavior as c, and a _fixing behavior as f. An example of a heterogeneous collaborative network is shown in Figure 4, where B_{new} is the newly reported software defect, developer D_3 is the reporter of the software defect, B_1 and B_2 belong to the historical software defects, and D_1 and D_2 are the developers involved. At the same time, developer D_3 also participated in commenting on historical software defect B_1 and historical software defect B_2. So far, the DARIP method has obtained a heterogeneous collaborative network built for software defect B_{new}.

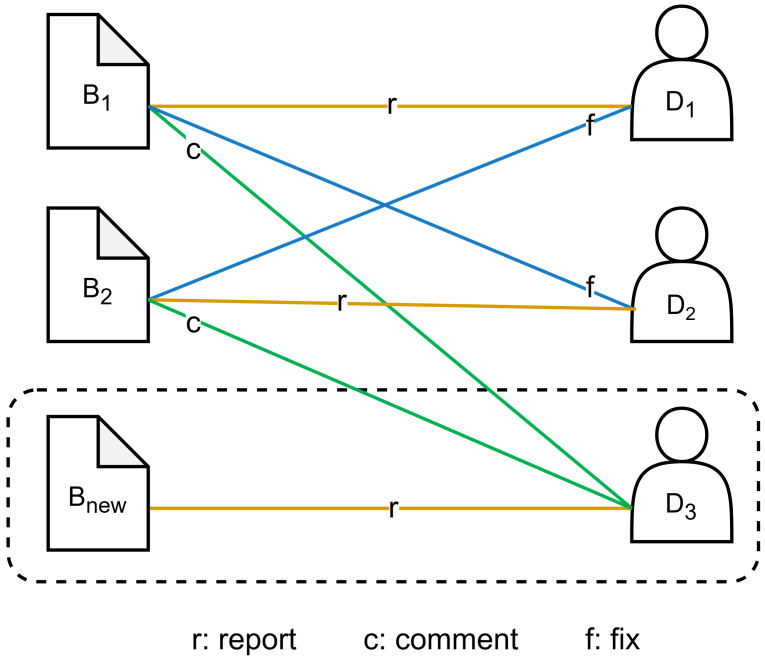

Figure 4. A sample of a heterogeneous collaborative network.

It should be noted that the heterogeneous collaborative network constructed by the DARIP method is undirected, because *D*-*c*-*B* and *B*-*c*-*D* illustrate the same scenario in the real software development process: the developer *D* comments on the software defect *B*. Since each software defect is reported at a different time, the historical software defects collected for each software defect are not the same. This makes the heterogeneous collaborative network built by each software defect unique. Therefore, this paper needs to build a separate heterogeneous collaborative network for each software defect.

Table 1. Definition of the heterogeneous collaborative network.

Structure	Type	Description
Node	Software Defect/Bug (B)	Software defects in a heterogeneous collaborative network include: current software defect and its historical software defects.
	Developer (D)	Developers in a heterogeneous collaborative network include: reporters, commentators, and fixers. Among them, the only developer involved in the current software defect is the reporter.
Edge	Report (r)	Developer report a software defect.
	Comment (c)	Developer comment on a software defect.
	Fix (f)	Developer fix a software defect.

4.3.2. Developer Characteristic Extraction

The heterogeneous collaborative network constructed for software defects is typical non-Euclidean spatial data. Therefore, it is necessary to select a suitable graph-embedding algorithm to extract the feature vectors of the reporter in current software defect from the heterogeneous collaborative network. Since the heterogeneous collaborative network constructed in this paper contains two types of nodes and three types of edges, it is crucial to select an algorithm that can handle heterogeneous networks. Dong et al. [47] proposed a graph-embedding algorithm, metapath2vec, which can deal with heterogeneous information networks, support custom meta-paths to guide the random wandering of nodes, and is able to obtain semantic and structural information between different nodes. Therefore, the metapath2vec algorithm is used in this paper to extract the reporter's feature vector in the current software defect from heterogeneous collaborative networks.

Extracting feature vectors of nodes from the network based on graph-embedding algorithm metapath2vec usually involves three steps: (1) Customising meta-paths. Meta-paths are paths formed by connecting different types of nodes in a heterogeneous network in a specific way. The metapath2vec algorithm supports researchers in fixing personalized problems by customizing meta-paths. (2) For a heterogeneous network, the metapath2vec algorithm guides random walks of nodes and obtains all node walk sequences based on metapath2vec. Considering the nodes in the network as words and the walk sequences of nodes as sentences, all possible node sequences form a corpus. (3) Feature vectors of nodes in the network are obtained by using the skip-gram model to extract word vectors in the corpus. According to the above steps, this paper needs to propose the custom meta-paths and then guide nodes in the heterogeneous collaborative network to walk around based on the custom meta-paths and obtain all possible walk sequences. Finally, all walk sequences are input into the skit-gram model, and the feature vector of the reporter in new software defect from the heterogeneous collaborative network are output.

(1) Custom meta-paths.

This paper defines the meta-paths based on the different collaborative relationships between developers. When constructing a heterogeneous collaborative network, we consider the three behaviors of developers: reporting, commenting, and fixing. According to the three development behaviors of developers, we can get six kinds of collaborative relationships among developers: report–report, report–comment, report–fix, comment–comment, comment–fix, and fix–fix. In this case, the report–report collaboration does not really exist because there is only one reporter for a software defect. Furthermore, the analysis of the dataset in Section 5 reveals that most software defects have only one fixer, and the percentage of defects with multiple fixers (two or more) is less than eight percent on average. Therefore, we ignore the collaborative relationship of fix–fix and finally obtain four types of collaborative relationships between developers and define the meta-paths. The meta-paths and the corresponding collaborative relationships are shown in Table 2.

Table 2. The meta-paths and collaborative relationships.

Meta-Paths	Collaborative Relationships
$D-f-B-r-D$ / $D-r-B-f-D$	A developer fixed a software defect reported by another developer.
$D-c-B-r-D$ / $D-r-B-c-D$	A developer commented on a software defect reported by another developer.
$D-c-B-f-D$ / $D-f-B-c-D$	A developer commented on a software defect fixed by another developer.
$D-c-B-c-D$	Both developers commented on the same software defect.

It is important to note that collaboration between developers does not take direction into account. For example, $D_1-f-B-r-D_2$ and $D_2-r-B-f-D_1$ illustrate the same scenario in the real software development process, where developer D_2 reported software defect B and developer D_1 fixed it.

(2) Guide random walks.

The defined meta-paths guide the nodes to walk randomly on the heterogeneous collaborative network, so that the node sequence containing the semantic information of the collaborative relationship can be obtained. Then, all the node sequences containing semantic information of collaborative relationships are inputted into the skip-gram model.

(3) Extract feature vectors.

The skip-gram model is a kind of Word2Vec model that predicts the context based on the central word. Specifically, it predicts other words before and after the word in the sliding window. Among them, the number of prediction words is determined by the size of the sliding window w. Thus, this paper essentially uses the skip-gram model to predict the other developers in a sequence of nodes that collaborate with a developer. For example, given a sequence of nodes as follows:

$$D_1 - c - B_1 - f - D_2 - f - B_2 - c - D_3 - c - B_3 - r - D_4 - f - B_4 - r - D_5 \quad (2)$$

Assuming that the central word is D_2, when the sliding window $w = 2$, the model can learn the set of developers $S_1 = \{D_1, D_3\}$ that has collaborative relationships with D_2. The collaborative distance between developer D_2 and developers D_1 and D_3 is 0. When the sliding window $w = 4$, the model can learn the set of developers $S_2 = \{D_1, D_3, D_4\}$ that has collaborative relationships with D_2. Among them, the collaborative distance between developer D_2 and developers D_1 and D_3 is 0, the collaborative distance between developer D_2 and developers D_4 is 1. When the sliding window $w = 6$, the model can learn the set of developers $S_3 = \{D_1, D_3, D_4, D_5\}$ that has collaborative relationships with D_2. Among them, the collaborative distance between developer D_2 and developers D_1 and D_3 is 0, the collaborative distance between developer D_2 and developers D_4 is 1, the collaborative distance between developer D_2 and developers D_5 is 2.

Finally, the skip-gram model outputs the feature vector of each node in the heterogeneous collaborative network. We take the feature vector of the reporter in the current software defect as the developer characteristic of the current software defect. According to the experimental results in Section 6.3, this paper sets the value of sliding window $w = 4$.

4.4. Classifier

In this paper, the DARIP method takes the fixers as the labels of software defects. Therefore, the task of assigning a developer to a software defect can be transformed into a classification task, and the developer assignment task can be solved by using the method of classification task.

In this paper, a fully connected neural network is used to build a classifier, which trains and learns the textual characteristics, developer characteristics, and labels input of software defects in the classifier and uses the ReLU function as the activation function. Since there are cases where there are multiple fixers for a software defect, i.e., multiple labels for a single sample, our classification task belongs to multi-label classification. For this purpose, for the last layer of full connectivity, we use the Sigmoid activation function to calculate the probability of assigning a software defect to each developer category. The mathematical formula of the Sigmoid function is shown in Formula (3),

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

In the field of mathematics, the Sigmoid function is monotonically increasing, derivable, and continuous. Its value domain $R(s) \in (0, 1)$. When the input value tends to negative infinity, the output value is close to 0. When the input value tends to positive infinity, the output value is close to 1. The mathematical properties of the Sigmoid function allow it to be used in neural networks as an activation function to normalize the output of each neuron. In addition, since the probability also takes values in the range of 0 to 1, the Sigmoid function is often applied to prediction probability models to solve classification problems. This paper uses the Sigmoid function to predict the probability of each label in the software defect separately [48], and the probability of different categories do not affect

each other. In the classifier, the formula for calculating the probability of assigning the i-th software defect to the j-th fixer is defined by Formulas (4)–(6),

$$\sigma(z_j) = \frac{1}{1 + e^{-z_j}} \tag{4}$$

$$\sigma(z_j) = p(dev_j|bug_i) \tag{5}$$

$$p(dev_j|bug_i) = \frac{1}{1 + e^{-z_j}} \tag{6}$$

where z_j is the ouput of the fully connected layer and the input of the Sigmoid function. Formula (4) is obtained by inserting $x = z_j$ into Formula (3). In Formula (5), bug_i is the i-th software defect, and dev_j is the j-th fixer. $\sigma(z_j)$ is expressed as the probability of assigning the i-th software defect to the j-th fixer. The final probability calculation Formula (6) can be obtained by substituting Formula (5) with Formula (4).

In addition, the number of software defects fixed by each developer varies, making the sample size different for different developer types, which may lead to a category imbalance in the dataset. This paper addresses the problem of category imbalance by using data augmentation technique [49], which extends the dataset by generating new training data from existing data, effectively reducing model overfitting. Specifically, we first delete the categories with a sample size of 1 in the dataset because too few samples will affect the training effect of the model. Secondly, the sample size of the largest category in the dataset is denoted as $maxNum$. Then, the formula $Threshold = p * maxNum$ for the small number of samples is proposed by setting the parameter $p = 0.8$. For each category, when the sample size of the category is less than $Threshold$, a sample is randomly selected from the original dataset of the category, and the original textual content in the sample is randomly exchanged to generate new textual content. For the newly generated text, the same processing method is used to obtain the textual characteristic of the sample. At the same time, we keep the developer characteristic unchanged and finally get the characteristic vector of the new sample. When the sample size of the category is not less than $Threshold$, the data augmentation process for the current category is stopped. Finally, the problem of category imbalance in the dataset is solved by traversing and processing each category.

We evaluate the performance of the model by splitting the training set and the test set and dividing them by a ratio of 8:2. During the training phase, we obtain the augmented training set based on the above data-augmentation method. We construct the cross-entropy loss function, train the classifiers based on the gradient descent optimization algorithm, use the dropout strategy to prevent the overfitting phenomenon, and finally generate the assignment model. During the test phase, the assignment model generated in the training phase is used to recommend appropriate developers for each software defect based on the data in the test set.

5. Dataset

In order to study the assignment of software defects, this paper needs to collect data on software defects and construct a dataset.

In order to collect software defects, this paper needs to select suitable open-source projects from the GitHub platform. First, we sorted projects in GitHub from highest to lowest based on the number of stars in order to select some popular projects. For the Top-50 open-source projects with the highest number of stars, we further filtered out projects that shared documentation or books and only considered projects that involved software development work. Then, for such projects, we counted the number of software defects in each project from inception until 2023. Since GitHub supports developers to identify features for reported issues by adding labels, we manually checked the label described as a bug in each software project and selected the issue with that label as a software defect. In order to ensure that software defects do not change in the future, this paper only considers

software defects whose status is closed. At the same time, in order to ensure that the collected software defects can be used for the training and learning of the assignment model, we selected the software defects with fixers. For the software defects that meet the above conditions, we filtered out the projects with a number of software defects that were less than 500 to ensure that there were enough software defects in each software project. Finally, we selected the Top-9 open-source software projects with a high number of stars and more than 500 software defects. The details of the 9 open-source projects are shown in Table 3. In Table 3, the number of software defects in the 9 open-source projects reached 20,280, using programming languages such as C++, C#, JavaScript, Go, and other mainstream programming languages.

For the software defects in nine open-source projects, we used GitHub Rest API to extract the title, body, reporter, commentator, and fixer information of each defect. At the same time, we collected the reporting time and closing time of the software defects. We define other software defects whose closing time is earlier than the reporting time of the current defect as their historical software defects. Some software defects may be assigned to virtual bots (users registered on the platform with a "bot" string in their name) instead of real developers. Since no real developers were used, these software defects were not useful for the training and learning the automated developer assignment task. Therefore, we removed these software defects assigned to virtual bots based on the "bot" string contained in the registered name. At the same time, we also removed the virtual bots when collecting developers. In addition, there are a small number of cases where the reporters of software defects are also their fixers. In order to better study the developer assignment method, we treated these software defects as follows: (1) If there was only one fixer and the fixer was also the reporter, the software defect was deleted directly. (2) If the software defect had more than one fixer, the defect was retained, and the fixer, who was also the reporter, was deleted from the fixer list.

Table 3. Information of nine open-source projects.

Open-Source Project	Period	Number of Stars	Label of Defects	Number of Defects	Number of Fixers	Main Programming Language
tensorflow/tensorflow	2015/11–2022/12	170k	type:bug	7208	341	C++
flutter/flutter	2015/11–2022/12	148 k	P4	741	143	Dart
electron/electron	2014/05–2022/12	105 k	bug	755	35	C++
vercel/next.js	2016/10–2022/12	97.9 k	kind: bug	592	33	JavaScript
kubernetes/kubernetes	2014/06–2022/12	94.5 k	kind/bug	3768	778	Go
microsoft/TypeScript	2014/07–2022/12	87 k	Bug	4601	51	TypeScript
microsoft/terminal	2017/10–2022/12	86.7 k	Issue-Bug	577	30	C++
microsoft/PowerToys	2019/09–2022/12	83.7 k	Issue-Bug	973	56	C#
ant-design/ant-design	2015/07–2022/12	83.5 k	Bug	1065	44	TypeScript

In addition, software defects may have one or multiple fixers. We measured the percentage of software bugs with multiple fixers across 9 projects and found that only 7.57%

of software defects had two or more fixers, and 92.43% of software defects had only one fixer. Therefore, when we customized the meta-paths, we did not consider the collaboration of two developers fixing a software defect at the same time.

6. Experimental Evaluation

6.1. Evaluation Metrics

For the developer assignment method DARIP, this paper evaluates the DARIP method by using two metrics: Mean Reciprocal Rank(MRR) [50] and Recall@k [51].

(1) Mean Reciprocal Rank (MRR)

The Mean Reciprocal Rank is a widely used evaluation metric in recommendation algorithms. It represents the average reciprocal ranking of the real fixers in the assignment list. The higher the ranking of a real fixer in the assignment list, the higher the average reciprocal ranking. If there are multiple real fixers, the one with the highest ranking is selected for the calculation. In this paper, we denote N as the number of software defects, and $rank(i)$ as the rank of the real fixer in the assignment list for the i-th software defect. The formula for calculating the Mean Reciprocal Rank is as follows:

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank(i)} \tag{7}$$

(2) Recall@k

The k indicates a configurable option. The *Recall@k* is the number of real fixers in the top-k ranking divided by the number of real fixers. The research goal of this paper is to assign appropriate fixers to software defects, so we hope to find out the real fixers of software defects as much as possible, so we use the *Recall@k* to evaluate the ability of the model to predict the fixers. In this paper, we denote N as the number of software defects, $real(i)_k$ as the number of real fixers in the top-k ranking of the assignment list for the i-th software defect, and $real(i)_{all}$ as the number of all real fixers for the i-th software defect. Finally, the average of the *Recall@k* of N software defects is calculated. The calculation formula is as follows:

$$Recall@k = \frac{1}{N}\sum_{i=1}^{N}\frac{real(i)_k}{real(i)_{all}} \tag{8}$$

In addition, in order to compare the difference in the assignment effect of different methods, this paper defines the gain value to calculate the difference in the assignment effect of method M_1 and method M_2, including the gain value of MRR and the gain value of *Recall@k*. The definitions are as follows:

$$Gain(MRR) = \frac{MRR(M_1) - MRR(M_2)}{MRR(M_2)} \tag{9}$$

$$Gain(Recall@k) = \frac{Recall@k(M_1) - Recall@k(M_2)}{Recall@k(M_2)} \tag{10}$$

6.2. Research Questions

This paper studies the following three questions:

RQ1: How is the performance of the DARIP method in assigning developers for software defects?

In this paper, we compare the DARIP method with the Multi-triage method proposed by Aung et al. [14] and validate the effect of the DARIP method of developer assignment for software defects in the datasets of nine open-source projects, respectively.

RQ2: Does the combination of different characteristics improve the assignment effect of the DARIP method?

The automatic developer assignment method proposed in this paper considers the textual and developer characteristics of software defects, respectively. In order to evaluate the necessity of selecting the above characteristics, this paper compares the assignment effect of different characteristic combinations and selects the appropriate characteristic combinations to apply to the DARIP method.

RQ3: How do different parameter settings affect the assignment effect of the DARIP method?

Different parameters are involved in the developer assignment method DARIP designed in this paper. We predict Top-N potentially related issues for software defects when extracting textual characteristics. We use the skip-gram model with a specified value of sliding window w to learn the feature representations of developers in heterogeneous collaborative networks. In order to evaluate the effect of different parameter settings on the developer assignment effect, this paper compares the assignment effect under different parameter values and selects the appropriate parameter size for the DARIP method.

6.3. Evaluation Results

6.3.1. RQ1: How Is the Performance of the DARIP Method in Assigning Developers for Software Defects?

Aung et al. [11] proposed multi-triage, a multi-task learning multi-classification method, which extracted the feature representation of bug descriptions and code snippets using a text encoder and an AST encoder, and improved the performance of the model by jointly learning the developer assignment and label classification tasks. In this paper, the Multi-triage method is used as a comparison method.

The experimental results are shown in Table 4. The average values of the Recall@5, the Recall@10, and the MRR of the method DARIP in 9 projects are 0.5902, 0.7160, and 0.4136, respectively. It can be seen that the assignment method DARIP in this paper outperforms the Multi-triage method in the Recall@5, the Recall@10, and the MRR.

Table 4. Definition of the heterogeneous collaborative network.

Project	Method	Recall@5	Recall@10	MRR
ant-design	DARIP	**0.9156**	**0.9778**	**0.6708**
	Multi-traige	0.7426	0.7919	0.5741
electron	DARIP	**0.7319**	**0.8795**	**0.5784**
	Multi-traige	0.7219	0.8278	0.5699
flutter	DARIP	0.2000	**0.3125**	**0.1412**
	Multi-traige	**0.2569**	0.2926	0.1306
kubernetes	DARIP	0.1271	0.1766	**0.1073**
	Multi-traige	**0.1286**	**0.2084**	0.0998
next.js	DARIP	**0.8202**	**0.9071**	**0.5997**
	Multi-traige	0.3517	0.6695	0.2397
PowerToys	DARIP	**0.6501**	**0.8347**	**0.4284**
	Multi-traige	0.5155	0.6443	0.4020
tensorflow	DARIP	**0.2980**	**0.4344**	**0.1770**
	Multi-traige	0.1446	0.3174	0.1333
terminal	DARIP	**0.8281**	**0.9676**	**0.5439**
	Multi-traige	0.5725	0.7812	0.4174
TypeScript	DARIP	**0.7410**	**0.9542**	**0.4757**
	Multi-traige	0.6163	0.7750	0.4004
Average	DARIP	**0.5902**	**0.7160**	**0.4136**
	Multi-traige	0.4501	0.5898	0.3297

Then, the gains of the DARIP method over the Multi-triage method are calculated on 9 projects for the two evaluation metrics, as shown in Table 5. Compared with the existing method Multi-triage, the average values of the Recall@5, the Recall@10, and the MRR of the DARIP method in 9 projects are increased by 31.13%, 21.40%, and 25.45%, respectively.

Table 5. The gain values of the DARIP method relative to the Multi-triage method.

Project	Gain(%)		
	Recall@5	**Recall@10**	**MRR**
ant-design	23.30%	23.48%	16.84%
electron	1.39%	6.25%	1.49%
flutter	−22.15%	6.80%	8.12%
kubernetes	−1.17%	−15.26%	7.52%
next.js	133.21%	35.49%	150.19%
PowerToys	26.11%	29.55%	6.57%
tensorflow	106.09%	36.86%	32.78%
terminal	44.65%	23.86%	30.31%
TypeScript	20.23%	23.12%	18.81%
Average	31.13%	21.40%	25.45%

6.3.2. RQ2: Does the Combination of Different Features Improve the Assignment Effect of the DARIP Method?

The assignment method proposed in this paper considers the textual and developer characteristics of software defects, respectively. In order to evaluate the necessity of selecting the above characteristics, this paper compares the assignment effect of different characteristic combinations and selects the appropriate characteristic combinations to apply to the DARIP method.

We remove textual characteristics and developer characteristics, respectively, and compare the differences in the assignment effect between the method of removing textual characteristic, the method of removing developer characteristic, and the DARIP method, as shown in Table 6. The results show that the DARIP method has higher Recall@5, Recall@10, and MRR on nine projects than the methods with textual characteristic removed and developer characteristic removed. This shows that considering both textual and developer characteristics can help improve the effect of developer assignment.

In conclusion, compared with the method of removing textual characteristic and the method of removing developer characteristic, the DARIP method shows a better assignment effect on nine projects. This fully demonstrates the importance of selecting the above characteristics in this paper.

Table 6. Comparison of different characteristics.

Project	Experimental Group	Recall@5	Recall@10	MRR
ant-design	DARIP	**0.9156**	**0.9778**	**0.6708**
	Remove Text	0.8828	0.9294	0.5418
	Remove Developer	0.8891	0.9682	0.6462
electron	DARIP	**0.7319**	**0.8795**	**0.5784**
	Remove Text	0.6632	0.8160	0.5477
	Remove Developer	0.6899	0.8501	0.5636
flutter	DARIP	**0.2**	**0.3125**	**0.1412**
	Remove Text	0.1761	0.2689	0.1298
	Remove Developer	0.1714	0.2642	0.1294

Table 6. *Cont.*

Project	Experimental Group	Recall@5	Recall@10	MRR
kubernetes	DARIP	**0.1271**	**0.1766**	**0.1073**
	Remove Text	0.1136	0.1655	0.1005
	Remove Developer	0.1253	0.1725	0.0980
next.js	DARIP	**0.8202**	**0.9071**	**0.5997**
	Remove Text	0.7598	0.8993	0.5928
	Remove Developer	0.8046	0.9013	0.5961
PowerToys	DARIP	**0.6501**	**0.8347**	**0.4284**
	Remove Text	0.6275	0.7264	0.3344
	Remove Developer	0.6359	0.8002	0.4220
tensorflow	DARIP	**0.2980**	**0.4344**	**0.1770**
	Remove Text	0.2275	0.3893	0.1402
	Remove Developer	0.2764	0.4250	0.1734
terminal	DARIP	**0.8281**	**0.9676**	**0.5439**
	Remove Text	0.7016	0.9656	0.4303
	Remove Developer	0.8221	0.9656	0.5414
TypeScript	DARIP	**0.7410**	**0.9542**	**0.4757**
	Remove Text	0.5089	0.8407	0.3271
	Remove Developer	0.7304	0.9404	0.4724

6.3.3. RQ3: How Do Different Parameter Settings Affect the Assignment Effect of the DARIP Method?

Different parameters are involved in the developer assignment method DARIP designed in this paper. This section analyzes the parameter settings of the number of predicted related issues N and the sliding window w in the skip-gram model.

When extracting the textual characteristics of software defects, we predict Top-N potential related issues for each software defect, weight them based on the cosine similarity between software defects and related issues, and finally get the textual characteristics of each software defect. If the parameter N takes different values, the resulting textual characteristics also have certain differences. We set the parameter N to 0, 1, 3, and 5, respectively, and compare the difference in the assignment effect when N takes different values, as shown in Table 7. Among them, $N = 0$ means that we do not consider the related issues and only extract textual characteristics based on the text information of software defects.

Table 7. Comparison of different characteristics.

Project	Top-N	Recall@5	Recall@10	MRR
ant-design	Without Issues	0.8989	0.9689	0.6451
	Top-1	**0.9156**	**0.9778**	**0.6708**
	Top-3	0.8891	0.9597	0.6507
	Top-5	0.8859	0.9640	0.6415
electron	Without Issues	0.7064	0.8716	0.5738
	Top-1	**0.7319**	**0.8795**	**0.5784**
	Top-3	0.6914	0.8501	0.5606
	Top-5	0.6751	0.8472	0.5552
flutter	Without Issues	0.1897	0.2773	0.1360
	Top-1	**0.2**	**0.3125**	**0.1412**
	Top-3	0.1811	0.3062	0.1323
	Top-5	0.1748	0.3046	0.1279

Table 7. *Cont.*

Project	Top-N	Recall@5	Recall@10	MRR
kubernetes	Without Issues	0.1261	0.1638	0.0911
	Top-1	**0.1271**	**0.1766**	**0.1073**
	Top-3	0.1015	0.1575	0.0976
	Top-5	0.1184	0.1572	0.0875
next.js	Without Issues	0.7549	0.8961	0.5677
	Top-1	**0.8202**	**0.9071**	**0.5997**
	Top-3	0.7851	0.8974	0.5698
	Top-5	0.7618	0.8916	0.5844
PowerToys	Without Issues	0.6376	0.8259	0.4202
	Top-1	**0.6501**	**0.8347**	**0.4284**
	Top-3	0.6085	0.7787	0.3865
	Top-5	0.6049	0.7859	0.3702
tensorflow	Without Issues	0.2937	0.4025	0.1698
	Top-1	**0.2980**	**0.4344**	**0.1770**
	Top-3	0.2773	0.3899	0.1598
	Top-5	0.2836	0.4105	0.1680
terminal	Without Issues	0.7916	0.9652	0.5345
	Top-1	**0.8281**	**0.9676**	**0.5439**
	Top-3	0.8221	0.9834	0.5371
	Top-5	0.8004	0.9755	0.5394
TypeScript	Without Issues	0.7119	0.9535	0.4519
	Top-1	**0.7410**	**0.9542**	**0.4757**
	Top-3	0.7286	0.9487	0.4666
	Top-5	0.7195	0.9509	0.4560

When considering related issues, it can be found that the Recall@5, the Recall@10, and the MRR when N is 1 are better than the Recall@5, the Recall@10, and the MRR when N is 3 and 5. This indicates that the method considering Top-1 related issues has better assignment effect than the methods considering Top-3 and Top-5 related issues. Moreover, with the increase of N, the assignment effect of the DARIP method decreases.

In the case of not considering related issues, it can be found that the Recall@5, the Recall@10, and the MRR when N takes the value of 1 are better than the Recall@5, the Recall@10, and the MRR without related issues on 9 projects. This suggests that considering the text information of related issues can help to improve the developers' assignment effect.

In conclusion, the DARIP method is the most effective when predicting the Top-1 potentially related issues for software defects. Therefore, in this paper, we set the value of N to 1.

At the same time, this paper uses the skip-gram model to extract developer characteristics from the walk sequences. The size of the sliding window w in the skip-gram model limits the collaborative distance between developers that can be learned. When w takes different values, the developer characteristics extracted from software defects are also different.

We set the sliding window w to 2, 4, and 6, respectively, and compare the differences in the assignment effect when w is set to different values, as shown in Table 8. The results show that the Recall@5, the Recall@10, and the MRR of the DARIP method when w is set to 4 are better than that of the methods when w is set to 2 and 6. Therefore, we set the value of w to 4 in this paper.

Table 8. Comparison of different characteristics.

Project	w	Recall@5	Recall@10	MRR
ant-design	2	0.8649	0.9564	0.5979
	4	**0.9156**	**0.9778**	**0.6708**
	6	0.8992	0.9682	0.6544
electron	2	0.7018	0.8532	0.5656
	4	**0.7319**	**0.8795**	**0.5784**
	6	0.6815	0.8408	0.5476
flutter	2	0.1968	0.3043	0.1353
	4	**0.2**	**0.3125**	**0.1412**
	6	0.1841	0.2991	0.1326
kubernetes	2	0.1254	0.1188	0.0888
	4	**0.1271**	**0.1766**	**0.1073**
	6	0.1229	0.1745	0.0978
next.js	2	0.7949	0.8976	0.5645
	4	**0.8202**	**0.9071**	**0.5997**
	6	0.8194	0.8863	0.5435
PowerToys	2	0.6199	0.7878	0.3946
	4	**0.6501**	**0.8347**	**0.4284**
	6	0.6206	0.7892	0.4050
tensorflow	2	0.2860	**0.4554**	0.1685
	4	**0.2980**	0.4344	**0.1770**
	6	0.2892	0.4298	0.1625
terminal	2	0.8180	0.9660	0.5274
	4	**0.8281**	**0.9676**	**0.5439**
	6	0.8101	0.9556	0.5395
TypeScript	2	0.6930	0.8995	0.4393
	4	**0.7410**	**0.9542**	**0.4757**
	6	0.6951	0.8953	0.4348

7. Discussion

In this paper, we propose the developer assignment method DARIP, which assigns appropriate developers to software defects by considering the prediction of related issues and the collaborative relationship between developers. Experimental results validate the effectiveness of the DARIP method in several ways.

The DARIP method takes into account the possible correlation between software defects and other issues, and mines additional valuable text information for software defects by predicting related issues. As shown in Table 7, the Recall@5, the Recall@10, and the MRR when N is 1 are better than the Recall@5, the Recall@10, and the MRR when N is 0. This indicates that the method considering Top-1 related issues has better assignment effect than the method without considering related issues. However, we find that the assignment effect of the DARIP method decreases with the increase of N. In the ant-design project, the Recall@5, the Recall@10, and the MRR when N is 5 are lower than the Recall@5, the Recall@10, and the MRR when N is 0. We analyze the difference of the assignment effect when N takes different values, and find that it is optimal when predicting Top-1 related issues for software defects.

Meanwhile, the DARIP method considers four types of collaborative relationships between developers: report–comment, report–fix, comment–comment, and comment–fix, and uses graph-embedding algorithm to learn collaboration between developers from the constructed heterogeneous collaborative network. We analyze the difference in assignment effect with different values of sliding window w, as shown in Table 8. The Recall@5, the Recall@10, and the MRR of the DARIP method when w is set to 4 are better than that of the method when w is set to 2. This indicates that it is not sufficient to consider collaborative

relationships where the collaborative distance between developers is 0 (direct collaboration, where two developers collaborate on the same software defect) but also collaborative relationships where the collaborative distance between developers is 1 (indirect collaboration, where both developers have a direct collaborative relationship with the same developer). However, the Recall@5, the Recall@10, and the MRR of the DARIP method when w is set to 4 are better than that of the method when w is set to 6. This indicates that the assignment effect decreases when considering collaborative relationships with greater distances. Therefore, when assigning developers for software defects, direct and indirect collaborative relationships between developers need to be considered, and collaborative relationships with greater distances are not recommended.

8. Threats to Validity

Threats to construct validity is concerned with the suitability of experimental results to the concepts or theories behind the experiments. Firstly, we selected 9 popular projects from the GitHub platform to verify our method. The 9 open-source projects involve a variety of common programming languages and contain 20,280 software defects, which is representative. Future work can collect more open-source projects to validate our method. Then, we predict related issues for a software defect from its historical software defects. These historical software defects belong to the same project as the software defect. However, when Li et al. [29] studied the correlation between issues, they found that the related issues of an issue in a project may originate from other projects on the GitHub platform. In future work, we will expand the data sources of related issues to evaluate the DARIP method.

Threats to internal validity is the degree to which research establishes a credible causal relationship between disposition and outcome. First, we investigate the assignment effect of the method based on the standard answer of the assignee to the software defect as the fixer. However, the real fixer may not be the best developer to fix the software defect, and other developers may also be capable of fixing it. Future work can manually label the developers who are capable of fixing software defects to construct the dataset. Then, this paper uses cosine similarity to calculate the correlation between software defects and related issues. In future work, we try to use different text similarity calculation methods and analyze the impact of using different similarity calculation methods on developer assignment methods.

Threats to external validity have to do with the universality of our research. This paper selects the GitHub platform to study the developer assignment of software defects but is uncertain about the effect of this method on other open-source platforms. Future work can collect data from more platforms to validate the DARIP method. Meanwhile, this paper investigates the developer assignment method for software defects in the Issue-Tracking System. However, the issues recorded in the Issue-Tracking System include not only software defects but also functional requirements proposed by developers. Such issues may also be assigned to developers to complete software development tasks. In future work, we consider extending the DARIP method to assign developers to all issues in the Issue-Tracking System.

9. Conclusions

In this paper, we propose a developer assignment method, DARIP, for open-source software defects. First, the DARIP method uses the BERT model to extract the textual vectors from software defects and all its historical software defects. Based on the cosine similarity, Top-N potential related issues are predicted for each software defect from all the historical software defects, and takes the text information of related issues together with the text information of software defects as textual input. The textual characteristics of software defects are obtained by comprehensively calculating the text vector of software defects and related issues. Secondly, a heterogeneous collaborative network is constructed based on the three development behaviors of developers: reporting, commenting, and fixing. The meta-paths are defined based on the four collaboration relationships between developers:

report–comment, report–fix, comment–comment, and comment–fix. The graph-embedding algorithm metapath2vec is used to extract developer characteristics from the heterogeneous collaborative network. Then, a classifier based on a deep learning model calculates the probability assigned to each developer category. Finally, the assignment list is obtained according to the probability ranking.

This paper verifies the performance of the DARIP method on 20,280 software defects in 9 popular projects. The experimental results show that the average values of the Recall@5, the Recall@10, and the MRR of the DARIP method in 9 projects are 0.5902, 0.7160, and 0.4136, respectively. Compared with the existing method Multi-triage, the average values of the Recall@5, the Recall@10, and the MRR of the DARIP method in 9 projects are increased by 31.13%, 21.40%, and 25.45%, respectively.

Author Contributions: Conceptualization, B.L. and J.J.; methodology, B.L. and J.J.; validation, B.L. and Z.L.; formal analysis, B.L.; investigation, B.L.; resources, L.Z. and J.J.; data curation, B.L.; writing—original draft preparation, B.L.; writing—review and editing, J.J.; supervision, L.Z.; project administration, J.J.; funding acquisition, L.Z. and J.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under grant No. 62177003.

Data Availability Statement: Data are contained within the article.

Acknowledgments: This research was supported by the National Natural Science Foundation of China.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Dabbish, L.; Stuart, C.; Tsay, J.; Herbsleb, J. Social coding in GitHub: Transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, Seatle, WA, USA, 11–15 February 2012; pp. 1277–1286.
2. Lima, A.; Rossi, L.; Musolesi, M. Coding together at scale: GitHub as a collaborative social network. In Proceedings of the International AAAI Conference on Web and Social Media, Oxford, UK, 27–29 May 2014; Volume 8, pp. 295–304.
3. Yang, B.; Yu, Q.; Zhang, W.; Wu, J.; Liu, C. Influence Factors Correlation Analysis in GitHub Open Source Software Development Process. *J. Softw.* **2017**, *28*, 1330–1342.
4. Bissyandé, T.F.; Lo, D.; Jiang, L.; Réveillere, L.; Klein, J.; Le Traon, Y. Got issues? who cares about it? a large scale investigation of issue trackers from github. In Proceedings of the 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), Pasadena, CA, USA, 4–7 November 2013; pp. 188–197.
5. Bertram, D.; Voida, A.; Greenberg, S.; Walker, R. Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, Savannah, GA, USA, 6–10 February 2010; pp. 291–300.
6. Xu, C.; Cheung, S.C.; Ma, X.; Cao, C.; Lu, J. Adam: Identifying defects in context-aware adaptation. *J. Syst. Softw.* **2012**, *85*, 2812–2828. [CrossRef]
7. Yan, A.; Zhong, H.; Song, D.; Jia, L. How do programmers fix bugs as workarounds? An empirical study on Apache projects. *Empir. Softw. Eng.* **2023**, *28*, 96–120. [CrossRef]
8. Wang, Y.; Chen, X.; Huang, Y.; Zhu, H.N.; Bian, J.; Zheng, Z. An empirical study on real bug fixes from solidity smart contract projects. *J. Syst. Softw.* **2023**, *204*, 96–120. [CrossRef]
9. Guo, S.; Zhang, X.; Yang, X.; Chen, R.; Guo, C.; Li, H.; Li, T. Developer activity motivated bug triaging: Via convolutional neural network. *Neural Process. Lett.* **2020**, *51*, 2589–2606. [CrossRef]
10. Shokripour, R.; Anvik, J.; Kasirun, Z.M.; Zamani, S. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR), San Francisco, CA, USA, 18–19 May 2013; pp. 2–11.
11. Jeong, G.; Kim, S.; Zimmermann, T. Improving bug triage with bug tossing graphs. In Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Amsterdam, The Netherlands, 24–28 August 2009; pp. 111–120.
12. Jahanshahi, H.; Cevik, M. S-DABT: Schedule and Dependency-aware Bug Triage in open-source bug tracking systems. *Inf. Softw. Technol.* **2022**, *151*, 107025. [CrossRef]

13. Yang, G.; Zhang, T.; Lee, B. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, Sweden, 21–25 July 2014; pp. 97–106.
14. Aung, T.W.W.; Wan, Y.; Huo, H.; Sui, Y. Multi-triage: A multi-task learning framework for bug triage. *J. Syst. Softw.* **2022**, *184*, 111133. [CrossRef]
15. Naguib, H.; Narayan, N.; Brügge, B.; Helal, D. Bug report assignee recommendation using activity profiles. In Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR), San Francisco, CA, USA, 18–19 May 2013; pp. 22–30.
16. Xia, X.; Lo, D.; Ding, Y.; Al-Kofahi, J.M.; Nguyen, T.N.; Wang, X. Improving automated bug triaging with specialized topic model. *IEEE Trans. Softw. Eng.* **2016**, *43*, 272–297. [CrossRef]
17. Zhang, W.; Cui, Y.; Yoshida, T. En-lda: An novel approach to automatic bug report assignment with entropy optimized latent dirichlet allocation. *Entropy* **2017**, *19*, 173. [CrossRef]
18. Bhattacharya, P.; Neamtiu, I. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In Proceedings of the 2010 IEEE International Conference on Software Maintenance, Timisoara, Romani, 12–18 September 2010.
19. Jonsson, L.; Borg, M.; Broman, D.; Sandahl, K.; Eldh, S.; Runeson, P. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empir. Softw. Eng.* **2016**, *21*, 1533–1578. [CrossRef]
20. Sarkar, A.; Rigby, P.C.; Bartalos, B. Improving bug triaging with high confidence predictions at ericsson. In Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 29 September–4 October 2019; pp. 81–91.
21. Lee, S.R.; Heo, M.J.; Lee, C.G.; Kim, M.; Jeong, G. Applying deep learning based automatic bug triager to industrial projects. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; pp. 926–931.
22. Mani, S.; Sankaran, A.; Aralikatte, R. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, Mumbai, India, 4–7 January 2019; pp. 171–179.
23. Canfora, G.; Ceccarelli, M.; Cerulo, L.; Di Penta, M. How long does a bug survive? An empirical study. In Proceedings of the 2011 18th Working Conference on Reverse Engineering, Limerick, Ireland, 17–20 October 2011; pp. 191–200.
24. Li, Z.; Tan, L.; Wang, X.; Lu, S.; Zhou, Y.; Zhai, C. Have things changed now? An empirical study of bug characteristics in modern open-source software. In Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, San Jose, CA, USA, 21 October 2006; pp. 25–33.
25. Almhana, R.; Kessentini, M. Considering dependencies between bug reports to improve bugs triage. *Autom. Softw. Eng.* **2021**, *28*, 1–26. [CrossRef]
26. Jahanshahi, H.; Chhabra, K.; Cevik, M.; Baþar, A. DABT: A dependency-aware bug triaging method. In Proceedings of the Evaluation and Assessment in Software Engineering , Trondheim, Norway, 21–23 June 2021; pp. 221–230.
27. Hong, Q.; Kim, S.; Cheung, S.C.; Bird, C. Understanding a developer social network and its evolution. In Proceedings of the 2011 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VA, USA, 25–30 September 2011; pp. 323–332.
28. Xuan, J.; Jiang, H.; Ren, Z.; Zou, W. Developer prioritization in bug repositories. In Proceedings of the 2012 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, 2–9 June 2012; pp. 25–35.
29. Wang, Q.; Xu, B.; Xia, X.; Wang, T.; Li, S. Duplicate pull request detection: When time matters. In Proceedings of the 11th Asia-Pacific Symposium on Internetware, Fukuoka, Japan, 28–29 October 2019; pp. 1–10.
30. Ma, W.; Chen, L.; Zhang, X.; Zhou, Y.; Xu, B. How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, 20–28 May 2017; pp. 381–392.
31. Zhang, Y.; Yu, Y.; Wang, H.; Vasilescu, B.; Filkov, V. Within-ecosystem issue linking: A large-scale study of rails. In Proceedings of the 7th International Workshop on Software Mining, Montpellier, France, 3 September 2018; pp. 12–19.
32. Li, L.; Ren, Z.; Li, X.; Zou, W.; Jiang, H. How are issue units linked? empirical study on the linking behavior in github. In Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Nara, Japan, 4–7 December 2018; pp. 386–395.
33. Liu, B.; Zhang, L.; Jiang, J.; Wang, L. A method for identifying references between projects in GitHub. *Sci. Comput. Program.* **2022**, *222*, 102858. [CrossRef]
34. Zhang, Y.; Wu, Y.; Wang, T.; Wang, H. iLinker: A novel approach for issue knowledge acquisition in GitHub projects. *World Wide Web* **2020**, *23*, 1589–1619. [CrossRef]
35. Liu, B.; Zhang, L.; Liu, Z.; Jiang, J. Cross-project Issue Recommendation Method for Open-source Software Defects. *J. Softw.* **2023**, 1–19. [CrossRef]
36. Zaidi, S.F.A.; Woo, H.; Lee, C.G. Toward an effective bug triage system using transformers to add new developers. *J. Sens.* **2022**, *2022*, 4347004 . [CrossRef]
37. Kim, M.H.; Wang, D.S.; Wang, S.T.; Park, S.H.; Lee, C.G. Improving the Robustness of the Bug Triage Model through Adversarial Training. In Proceedings of the International Conference on Information Networking, Jeju, Republic of Korean, 12–15 January 2022; pp. 478–481.

38. Zhang, W.; Zhao, J.; Wang, S. SusTriage: Sustainable Bug Triage with Multi-modal Ensemble Learning. In Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology, Melbourne, VIC, Australia, 14–17 December 2021; pp. 441–448.
39. Jahanshahi, H.; Cevik, M.; Mousavi, K.; Basar, A. ADPTriage: Approximate Dynamic Programming for Bug Triage. *Trans. Softw. Eng.* **2023**, *49*, 4594–4609. [CrossRef]
40. Banitaan, S.; Alenezi, M. Decoba: Utilizing developers communities in bug assignment. In Proceedings of the 2013 12th International Conference on Machine Learning and Applications, Miami, FL, USA, 4–7 December 2013; Volume 2, pp. 66–71.
41. Zhang, W.; Wang, S.; Wang, Q. KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. *Inf. Softw. Technol.* **2016**, *70*, 68–84. [CrossRef]
42. Zaidi, S.F.A.; Lee, C.G. Learning graph representation of bug reports to triage bugs using graph convolution network. In Proceedings of the 2021 International Conference on Information Networking (ICOIN), Jeju, Republic of Korean, 13–16 January 2021; pp. 504–507.
43. Hu, H.; Zhang, H.; Xuan, J.; Sun, W. Effective bug triage based on historical bug-fix information. In Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering, Naples, Italy, 3–6 November 2014; pp. 122–132.
44. Yadav, A.; Singh, S.K.; Suri, J.S. Ranking of software developers based on expertise score for bug triaging. *Inf. Softw. Technol.* **2019**, *112*, 1–17. [CrossRef]
45. Su, Y.; Xing, Z.; Peng, X.; Xia, X.; Wang, C.; Xu, X.; Zhu, L. Reducing bug triaging confusion by learning from mistakes with a bug tossing knowledge graph. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 191–202.
46. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
47. Dong, Y.; Chawla, N.V.; Swami, A. metapath2vec: Scalable representation learning for heterogeneous networks. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, 13–17 August 2017; pp. 135–144.
48. Goel, P.; Kumar, S.S. Certain class of starlike functions associated with modified sigmoid function. *Bull. Malays. Math. Sci. Soc.* **2020**, *43*, 957–991. [CrossRef]
49. Wei, J.; Zou, K. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv* **2019**, arXiv:1901.11196.
50. Zhao, J.; Zhou, Z.; Guan, Z.; Zhao, W.; Ning, W.; Qiu, G.; He, X. Intentgc: A scalable graph convolution framework fusing heterogeneous information for recommendation. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 2347–2357.
51. Altosaar, J.; Ranganath, R.; Tansey, W. RankFromSets: Scalable set recommendation with optimal recall. *Stat* **2021**, *10*, e363. [CrossRef]