A time-based approach to automatic bug report assignment

Ramin Shokripour\textsuperscript{a,b}, John Anvik\textsuperscript{b}, Zarinah M. Kasirun\textsuperscript{a}, Sima Zamani\textsuperscript{a}

\textsuperscript{a} Faculty of Computer Science \& Information Technology, University of Malaya, Kuala Lumpur, Malaysia
\textsuperscript{b} Department of Computer Science, Central Washington University, Ellensburg, WA 98926, USA

**ABSTRACT**

Bug assignment is one of the important activities in bug triaging that aims to assign bugs to the appropriate developers for fixing. Many recommended automatic bug assignment approaches are based on text analysis methods such as machine learning and information retrieval methods. Most of these approaches use term-weighting techniques, such as term frequency-inverse document frequency (tf-idf), to determine the value of terms. However, the existing term-weighting techniques only deal with frequency of terms without considering the metadata associated with the terms that exist in software repositories. This paper aims to improve automatic bug assignment by using time-metadata in tf-idf (Time-tf-idf). In the Time-tf-idf technique, the recency of using the term by the developer is considered in determining the values of the developer expertise. An evaluation of the recommended automatic bug assignment approach that uses Time-tf-idf, called ABA-Time-tf-idf, was conducted on three open-source projects. The evaluation shows accuracy and mean reciprocal rank (MRR) improvements of up to 11.8% and 8.94%, respectively, in comparison to the use of tf-idf. Moreover, the ABA-Time-tf-idf approach outperforms the accuracy and MRR of commonly used approaches in automatic bug assignment by up to 45.52% and 55.54%, respectively. Consequently, consideration of time-metadata in term weighting reasonably leads to improvements in automatic bug assignment.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Bug triaging, an important process in software maintenance, has a significant effect on the quality of software projects (Jeong et al., 2009). In this process, a project member called the triager, who may also be one of the developers, investigates the validity of the reported bugs. Valid bugs are then assigned to the most appropriate developer(s) for fixing. The process of assigning the bugs to developers is called bug assignment. Traditional manual bug triaging is time-consuming and tedious. More importantly, bug triage imposes extra costs on projects (Anvik and Murphy, 2011). Accordingly, various research has sought to improve this process by using (semi)automatic bug assignment processes (e.g. Anvik and Murphy, 2011; Kagdi et al., 2012; Servant and Jones, 2012; Tamrawi et al., 2011).

Machine learning (ML) methods (Ahsan et al., 2009; Anvik and Murphy, 2011; Bhattacharya et al., 2012; Cubranic and Murphy, 2004; Nasim et al., 2011; Tamrawi et al., 2011) and information retrieval (IR) methods (Kagdi et al., 2012; Linares-Vasquez et al., 2012; Lucca, 2002; Matter et al., 2005; Moin and Neumann, 2012; Nagwani and Verma, 2012; Zhang and Lee, 2013) have been widely used by researchers in bug assignment approaches. These approaches use textual information which was extracted from software repositories, to assign developers to new bugs. Although recent non-text based approaches are emerging to address the limitations of text analytic methods (Panichella et al., 2013), text-based methods are still the most-used and effective techniques in this area (Sun et al., 2014).

Term similarities between a new bug report and information resources are used for establishing a relationship between the new bug report and activities of the developers. The importance of each term in establishing this relationship is determined using a term-weighting technique. The most commonly used term-weighting technique for automatic bug assignment is term frequency-inverse document frequency (tf-idf) (Cavalcanti et al., 2014). A significant difference between the textual resources of software projects and documents in other areas (e.g. newspapers) is the existence of metadata, specifically the time of using the term.

This paper presents ABA-Time-tf-idf, an automatic bug assignment approach using the Time-tf-idf term weighting technique. ABA-Time-tf-idf improves the accuracy of a bug assignment approach. Compared to other approaches, this new approach is lightweight, as it only includes a time-based term weighting technique and a simple ranking method to rank developers based on their expertise. To the best of our knowledge, this approach is unique. An evaluation against other commonly used bug assignment approaches across three different datasets shows that the ABA-Time-tf-idf approach obtains better performance than the comparable ML,
IR, and ABA-tf-idf approaches. ABA-Time-tf-idf outperforms the SVM, Naïve Bayes, VSM and SUM approaches by as much as 45%, 28%, 11% and 19%, respectively. The new approach also improves on the mean reciprocal rank (MRR) by up to 55.54%, 30.88%, 15.37% and 25.03%. Finally, ABA-Time-tf-idf also outperforms the accuracy and MRR of ABA-tf-idf by up to 11.8% and 8.94%.

The rest of this paper proceeds as follows. First, the motivation of this research is presented in Section 2. Next, the details of the proposed method are described in Section 3. The empirical evaluation of the method is presented in Section 4 and the evaluation results and analysis are presented and discussed in Section 5. Section 6 discusses the threats to validity. Then, in Section 7, the most related works in bug assignment are investigated. Finally, the paper is concluded in Section 8.

2. Motivation

Bug assignment approaches which use text analysis techniques use the term similarity between the new bug report and software information resources, such as bug repository and source code, to establish a relationship between the new bug report and developer expertise. Recently, researchers have augmented the text analysis portion of such approaches with metadata. Examples of the use of metadata include the product and component of the bug (Xia et al., 2013), bug tossing behavior (Bhattacharya et al., 2012; Jeong et al., 2009), developer communication social networks (Wu et al., 2011; Zhang and Lee, 2013), and source code locations of the bug (Kagdi et al., 2012; Linares-Vasquez et al., 2012; Servant and Jones, 2012). This means that the accuracy of these approaches depends on the accuracy of the text analysis technique. Therefore, improving the accuracy of the text analysis technique could result in improving the accuracy of the bug assignment approach. This paper presents a new bug assignment approach that uses the time when the terms were used by the developer in the term weighting step of the text analysis process.

Other approaches have also used time-metadata, such as the time spent fixing previous bugs (Nguyen et al., 2012) and the time distance between the last activity of developer and the reporting date of the new bug (Bhattacharya et al., 2012; Kagdi et al., 2012), to improve the results of the text analysis methods. However, time-metadata have been used by the other techniques, we used time-metadata to determine the weight of the terms used by the developers. The proposed term weighting technique in this paper is referred to as time-based tf-idf (Time-tf-idf). An advantage of this technique is that it removes the need for activity thresholding (Anvik and Murphy, 2011; Matter et al., 2009), whereby developers who have been inactive for more than the threshold are removed from the developer list. Such pruning of the data may result in the loss of useful information. Time-tf-idf addresses this issue by giving lower weights to the earlier activities, but not removing them. In other words, early activities which will have less effect in determining the expertise are given lower weights than recent activities, as recent activities will have a greater effect in determining developer expertise. This is due to the use of the time difference between the terms used in each activity and those used in a new bug report. The larger the time difference, the smaller the term weight for the terms in that activity.

Moreover, projects have different goals or address different requirements at different points in time (Gómez et al., 2009), and the term weights should reflect this situation. For example, a developer who worked on a feature of the project five months ago has more potential to resolve a new bug related to this feature than the developer who worked on the same feature two years ago. Therefore, the last time that a developer used a keyword associated with that feature is a suitable parameter for assessing the developer’s current expertise relative to the feature. Furthermore, developers change their expertise, such as moving from one component or product to another component or product. Accordingly, the vocabularies that the developers use also change to reflect changes in their expertise. Considering the time at which a developer uses terms may therefore improve developer recommendation accuracy.

3. Proposed approach

The proposed approach in this research seeks to improve automatic bug assignment by using the terms’ time-metadata as an effective parameter in weighting the terms in the tf-idf term-weighting technique (time-based tf-idf). The tf-idf technique only deals with the frequency of the terms in the document and corpus to determine their value. However, time-based tf-idf (Time-tf-idf) also considers the time of using the term in the project in determining the weight. Fig. 1 shows an overview of the recommended automatic bug assignment approach (ABA-Time-tf-idf approach), that uses the Time-tf-idf term weighting technique. The ABA-Time-tf-idf approach has three stages: corpus creation, expertise determination, and developer recommendation. The details of these steps are described in the following sections.

3.1. Corpus creation

Regarding the usage of the time-metadata in the proposed approach, the corpus creation step is different from the corresponding step in the commonly used text analysis methods. In ABA-Time-tf-idf, the required data is collected from the version control system (VCS), a software repository for managing changes to source code and other relevant project documents. From the source code, the identifiers are extracted and used to establish the relationship between a new bug report and developer activities in the project. Therefore, each identifier in the project’s source code is associated with a developer and each identifier-developer relationship represents a developer activity.

Source code identifiers are used to name the entities of the source code (such as files, classes, and methods). In this study, the names of classes, methods, fields, and parameters of the methods are extracted, as they have been found to contain more useful and less noisy data. Identifiers are often the concatenation of a set of terms used to indicate the functionality of the source code entity (Abebe and Tonella, 2010). Therefore, the extracted identifiers are also decomposed into their components using the identifier tokenization method recommended by Butler et al. (2011). Such an extracted identifier is hereafter referred to as a “decomposed identifier”. The method of Butler et al. first decomposes the identifiers using common rules of term decomposition. Examples of such rules are decomposing based on the internal capitalization and decomposing based on separator characters. In addition, their approach improves tokenization by increasing the accuracy of tokenizing single identifiers and cases where identifiers contain digits. This is done by using a set of heuristics and other methods, such as word recognition and a recursive algorithm for finding the project’s vocabulary in the identifiers. In this study, both the complete and decomposed identifiers are used in representing developer activity.

After extracting the identifiers, they need to be associated with the corresponding metadata. For each extracted identifier, the name of the developer who is associated with the identifier and the time stamp of the commit into the source code repository are extracted for use in term weighting. The commit time is used as the creation time of the identifier. Consequently, it is possible that a term which is used in more than one identifier of a file has various times associated with it in the generated dataset.

The results of Capobianco et al. (2013) and our recent investigation (Zamani et al., 2014) demonstrate that only using the noun terms of the decomposed identifiers significantly reduces the volume of the dataset and also the noisy data without decreasing the...
accuracy. Therefore, in this research only noun terms are used as the dataset. Furthermore, the terms that have less than three characters are filtered out. Next, the terms are lemmatized in order to reduce the different forms of a term (Shokripour et al., 2013). Lemmatization uses a vocabulary and morphological analysis of the terms to remove inflectional endings, and return the base or dictionary form of a term, or lemma (Capobianco et al., 2013; Manning et al., 2008). Finally, an index of terms is created where each unigram term is linked to one or more developers and to the last time the developer used the term. As the output of this step, a triplet index is created containing the extracted term, the associated developer and the last time the developer used the term.

3.2. Term-weighting

The tf-idf technique determines the importance of terms based on the frequency of term appearance in the document and corpus. The weight of a term ($t$) in a document ($d$) is calculated using Eq. (1) where $\text{TermFreq}(t,d)$ is the number of appearances of the term in the document. $\text{idf}$ is the ratio of the number of documents in the corpus ($N$) containing the term ($\text{DocFreq}(t)$), which forms the second expression of the tf-idf formula (Fieldman and Sanger, 2006).

$$\text{tf–idf}(t, d) = \text{TermFreq}(t, d) \times \log \left( \frac{N}{\text{DocFreq}(t)} \right) \quad (1)$$

Note that in the tf-idf calculation, the weight of a term is calculated without considering the time of using the term. For example, if two developers have used the same term for a feature of the project at two different times (e.g. five months ago and two years ago), tf-idf gives equal weights for both of the developers. However, knowledge can decay over time, so the developer who worked on the feature recently is more likely to recall more relevant information than the developer who worked on the feature two years ago. Moreover, lack of activity on a feature can lead to a lack of information about recent changes for the feature. Therefore, weighting the terms according to the time of last use can align the term-weighting technique with reality. Consequently, the last time that a developer used a feature's term is a suitable indication of the developer's current expertise for the feature. Hence, the last date of term usage by a developer is used as metadata in calculating the weight of the term for that developer. However, the terms used earlier by a developer should have lower weights. The inverse of the time distance between the reported date of the bug and last date of usage of the term by the developer can satisfy this condition.

In Time-tf-idf, the activity areas of the developers in various times of the project are considered implicitly. For example, when a developer uses the term(s) of a specific feature two years ago, but after that does not use the relevant term(s), it can be inferred that the developer is no longer active on this feature. By using the time-metadata for determining the weight of the terms, there is no need for a predetermined threshold to check the activity status of the developers as has been done in other approaches (Anvik and Murphy, 2011; Matter et al., 2009). Moreover, although threshold techniques could be considered to also use time-metadata, Time-tf-idf makes a finer grained use of time-metadata than threshold techniques.

The proposed time-based tf-idf technique (Time-tf-idf), which is used in the term weighting step of the ABA-Time-tf-idf approach, combines two expressions for calculating the weight of the terms (see Eq. (2)). The first expression $\text{tf-idf}(t, d)$ calculates the values of a term ($t$) used by a developer ($d$) using the tf-idf formula, and the second expression (Recency of use($t, d, B$)) determines the weight of the term ($t$) used by the developer based on the distance between the reporting date of the new bug report ($B$) and the last date of usage by the developer.

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$$

$$= \text{TermFreq}(t, d) \times \log \left( \frac{\#\text{ProjDev}}{\#\text{DevTermFreq}(t)} \right) \quad (2)$$

Fig. 1. An overview of the ABA-Time-tf-idf approach.
As mentioned above, Time-tf-idf uses the inverse of the time distance between the last date of term usage by a developer and the reporting date of the new bug report as the time-metadata. This leads to lower weights for the terms that have been used earlier. The role of the idf expression in the tf-idf formula gives a higher weight to the keywords of the project. Accordingly, keywords have a more prominent role in establishing the relationship between two documents. Thus, the Time-tf-idf term-weighting technique should give a higher weight to keywords, and the effect of the time parameter on the term weights should account for the importance of the term in the project. To achieve this goal, the recency of use (Eq. (3)) for each term is calculated as the inverse of the date difference between the reporting date of the new bug (BugDate(B)) and last date of using the term by the developer (DateDevTerm(d, t)), added to the inverse of the number of developers who have used the term in the project (#DevTermFreq(t)). This gives a higher weight to the keywords that had been used far in the past relative to non-keywords that are used at that same time. This means that for keywords with a low #DevTermFreq(t) (i.e. a small denominator for the first term of Eq. (3)), the term obtains a higher weight than for a term with a high #DevTermFreq(t) (i.e. large denominator for the first term of Eq. (3)). Therefore, the common terms (non-keywords) which were used long ago have less of an effect when establishing the relationship between the new bug report and the developer’s expertise, and the converse is also true — uncommon terms (i.e. keywords) have a more significant role in term weighting. Correspondingly, the recently used keywords, which have a small time difference with the reported bug, have a higher weight than the keywords which were used a long time ago.

Accordingly, due to the higher weights of the terms that had been used recently, they have a greater effect in determining the expertise of the developer. Since the earlier activities of the developer are not eliminated, the terms used in these activities still have a role in determining the developer expertise. This is a key difference between our proposed approach and approaches that use a threshold in determining the developer expertise.

Recency of use(t, d, B) = \frac{1}{#DevTermFreq(t)} \left( \frac{1}{\sqrt{\text{BugDate}(B) - \text{DateDevTerm}(d, t)}} \right) \tag{3}

Finally, the Time-tf-idf term-weighting technique calculates the weight of each term (t) used by each developer (d) to make a decision about a new bug report (B). This is done by multiplying the tf-idf(t, d) and Recency of use(t, d, B) expressions (Eq. (4)).

Time–tf–idf(t, d, B) = \text{tf–idf}(t, d) \times \text{Recency of use}(t, d, B) \tag{4}

3.3. Developers’ expertise calculation and recommendation

The calculated term weights for the developers are used to determine the developer expertise for fixing a new bug report. Prior bug assignment approaches used the obtained weights from the tf-idf technique in text analysis algorithms, such as converting the text to a vector in a vector space model (VSM) approach (see Section 4.1.3). The ABA-Time-tf-idf approach combines the time-based tf-idf term-weighting technique (Time-tf-idf technique) with a simple ranking method. In this way, ABA-Time-tf-idf directly uses the term-weighting results to determine the similarity between new bug reports and developer expertise. This removes the need for computations such as converting to a vector and calculating the cosine similarity to establish a relationship.

The ABA-Time-tf-idf approach uses a simple ranking method for determining the expertise of the developers and ranking the developers based on their expertise. In this step, expertise of each developer (d) for fixing a new bug (B) is the total weight of all the terms that appear in both the new bug report and the developer activities (Eq. (5)).

\text{Expertise}_{\text{Dev}}(d, B) = \sum_{i=1}^{#\text{CommonTerms}} \text{Time–tf–idf}(t_i, d) \tag{5}

To make a recommendation, the ranking method sorts the developers in a descending list based on the calculated expertise for the developers by Eq. (5). The developers at the top of this list are considered to have the highest expertise for fixing the new bug.

3.4. Example of usage

Having presented the details of the proposed approach, we present an example to further clarify how developer expertise is determined using ABA-Time-tf-idf compared to ABA-tf-idf.

In this example, bug, from the Eclipse project is used. This bug was fixed by developer “Maeschi”. Table 1 presents a subset of the common terms between the activities of three developers and the bug report, and the calculated values for the date difference, tf-idf weighting and Time-tf-idf weighting.

The first column lists the names of three project developers. The next column shows five terms that are common between the developer’s activity and the bug report. The next three columns show the number of dates used to calculate the activity, the date of issue of the bug report, and the calculated values for the date difference, tf-idf weighting, and Time-tf-idf weighting.

<table>
<thead>
<tr>
<th>Developer</th>
<th>Term</th>
<th>Date</th>
<th>Date issued</th>
<th>Date difference (days)</th>
<th>ABA-tf-idf</th>
<th>Recency of use</th>
<th>ABA-Time-tf-idf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pnuellet</td>
<td>Destination</td>
<td>2004-10-20</td>
<td>2001-06-06</td>
<td>1232</td>
<td>5.4931</td>
<td>0.1118</td>
<td>0.6143</td>
</tr>
<tr>
<td>Property</td>
<td>Tree</td>
<td>2004-10-20</td>
<td>2002-12-09</td>
<td>681</td>
<td>2.6881</td>
<td>0.0818</td>
<td>0.2199</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42.0341</td>
<td>6.0995</td>
<td></td>
</tr>
<tr>
<td>Maeschi</td>
<td>Destination</td>
<td>2004-10-20</td>
<td>2004-02-17</td>
<td>246</td>
<td>7.6903</td>
<td>0.1471</td>
<td>1.1312</td>
</tr>
<tr>
<td>Location</td>
<td>Part</td>
<td>2004-10-20</td>
<td>2004-10-08</td>
<td>12</td>
<td>17.2609</td>
<td>0.3257</td>
<td>5.6221</td>
</tr>
<tr>
<td>Property</td>
<td>Tree</td>
<td>2004-10-20</td>
<td>2004-07-08</td>
<td>104</td>
<td>9.4085</td>
<td>0.1286</td>
<td>1.2100</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76.5597</td>
<td>15.5188</td>
<td></td>
</tr>
<tr>
<td>Akiezn</td>
<td>Destination</td>
<td>2004-10-20</td>
<td>2003-08-19</td>
<td>428</td>
<td>90.0862</td>
<td>0.1317</td>
<td>11.8617</td>
</tr>
<tr>
<td>Location</td>
<td>Part</td>
<td>2004-10-20</td>
<td>2003-05-08</td>
<td>531</td>
<td>7.7674</td>
<td>0.0804</td>
<td>0.6248</td>
</tr>
<tr>
<td>Property</td>
<td>Tree</td>
<td>2004-10-20</td>
<td>2003-08-13</td>
<td>434</td>
<td>9.4085</td>
<td>0.0980</td>
<td>0.8170</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>128.8187</td>
<td>15.3518</td>
<td></td>
</tr>
</tbody>
</table>
reporting date for the bug report, the date that the developer last used the term, and the difference (in days) between these two dates, respectively. The next column shows the term weight using the traditional tf-idf technique in the ABA-tf-idf approach. This is followed by the recency-of-use value and the term weight using the Time-tf-idf technique in the ABA-Time-tf-idf approach.

From the table, we see that the use of only statistical information (i.e. the ABA-tf-idf approach) can result in choosing the incorrect developer (“Akiezun”) if the developer has a large amount of activity, regardless of when the activity occurred. However, by considering how recently the terms were used, the developer that most recently worked on parts related to the feature (see Date used column) can be determined (“Maeschli”, the actual fixer).

This example demonstrates how the consideration of time can positively affect the determination of developer expertise and result in a more accurate assignment recommendation.

4. Empirical evaluation

In this section, the setup for the empirical evaluation of the proposed approach is presented. The setup is organized based on the recommended method put forth by Wohlin et al. (2012).

This research seeks to answer the following two questions:

1. Does using the time-metadata in determining the weight of terms improve the accuracy of a bug assignment approach (RQ1)?
2. Does a lightweight bug assignment approach perform better than previous ML and IR automatic bug assignment approaches (RQ2)?

To answer RQ1 and evaluate the effectiveness of using a time-based term weighting technique, the ABA-Time-tf-idf approach is compared to an approach, called ABA-tf-idf, that uses the traditional tf-idf term weighting technique. The ABA-tf-idf approach, similar to ABA-Time-tf-idf, includes the tf-idf term weighting technique and uses the same simple method for ranking developers. The difference between the two approaches is the use of time-metadata in weighting the terms. To answer RQ2, the performance of the proposed bug assignment approach is compared to four different ML and IR approaches on three subject systems.

4.1. Context selection

In this section the different contexts that are necessary for evaluating and comparing the proposed approach are described. The method of selecting the subject systems on which the proposed approach and baseline approaches should be evaluated are described, as well as a brief description for each subject system. Moreover, the method of selecting the test set for subject systems is described in this section. Finally, the baseline approaches that will be considered in the evaluation are described briefly.

4.1.1. Subjects

We evaluated our approach using data collected from three open-source projects: Eclipse JDT, NetBeans, and ArgoUML projects. These subject systems, which have been used in previous bug assignment researches, are of different scales. This helps to investigate the effect of different dataset sizes on the performance of the ABA-Time-tf-idf approach. The JDT project is a collection of plug-ins for the Eclipse framework that provides the tools to implement a Java integrated development environment (IDE). NetBeans is an IDE for Java development, but it also supports other languages, such as PHP and C/C++. ArgoUML is a commonly used open source UML modeling tool. Some of the properties of the datasets for these subject systems are presented in Table 2.

4.1.2. Objectives

There are two general techniques for constructing the test sets when evaluating approaches for automatic bug assignment. For the first technique, the bug reports of the test set are randomly selected from the fixed bugs across the whole of the project’s lifetime. The second technique is to select the bug reports for the test set from a specific period of the project’s lifetime. Both techniques are used for constructing the test sets in this work.

The first series of test sets were randomly chosen from across each project’s lifetime. This makes the dataset independent of time, and hopefully unbiased. To perform the evaluation, five test sets, each of which includes 100 fixed bugs randomly selected from all of the fixed bugs in the project were used. In other words, the bug reports of the test sets belong to different periods of the project’s lifetime. Each of these bug reports were associated with the set of commits to the project’s version control systems (VCS) before their reporting date. This is done to accurately simulate the data that was available at the time when the bug was reported.

The second series of test sets were created to evaluate the approaches using data having different characteristics. The projects have different goals or address different requirements at different points in time (Gómez et al., 2009). Moreover, it is known that the most recent modifications to a project are most likely the cause of future bugs or defects (Hassan and Holt, 2005). Therefore, the bug reports of each period reflect the characteristics of the project in that time period. For this reason, two test sets were created from two different periods in the project’s lifetime for each project. Specifically the first 100 fixed reports for which the fixing developer could be determined formed an ‘early’ test set and similarly the most recent 100 fixed reports, at the time of data collection, formed a ‘late’ test set. The IDs of the used bugs in test sets of the projects are available online.

The datasets used for training all of the approaches when testing with the Early and Late testing sets, consisted of source code and corresponding developer information from the time periods covered by the testing sets. In other words, for the early training set, the training data consisted of the extracted data from the source code of

\[\text{Table 2} \]

Properties of the extracted data from artifacts of the subject systems.

<table>
<thead>
<tr>
<th></th>
<th>Eclipse</th>
<th>NetBeans</th>
<th>ArgoUML</th>
</tr>
</thead>
<tbody>
<tr>
<td>First commit</td>
<td>2001-05-03</td>
<td>1999-02-04</td>
<td>1998-01-27</td>
</tr>
<tr>
<td>Last commit</td>
<td>2011-12-15</td>
<td>2010-06-25</td>
<td>2012-03-14</td>
</tr>
<tr>
<td># Developers</td>
<td>58</td>
<td>175</td>
<td>40</td>
</tr>
<tr>
<td># Java files</td>
<td>8308</td>
<td>1375</td>
<td>4371</td>
</tr>
<tr>
<td># Commits</td>
<td>162,321</td>
<td>67,216</td>
<td>58,874</td>
</tr>
<tr>
<td>Avg. number of commits per day</td>
<td>48.8</td>
<td>16.36</td>
<td>11.40</td>
</tr>
<tr>
<td># reported bugs</td>
<td>47,265</td>
<td>185,578</td>
<td>6413</td>
</tr>
<tr>
<td>Avg. number of bugs per day</td>
<td>12</td>
<td>44.62</td>
<td>1.24</td>
</tr>
<tr>
<td># Fixed bugs</td>
<td>21,466</td>
<td>69,651</td>
<td>2807</td>
</tr>
<tr>
<td>Avg. number of fixed bugs per day</td>
<td>5.5</td>
<td>16.74</td>
<td>0.54</td>
</tr>
</tbody>
</table>

1 A software repository for managing changes to source code and other relevant project documents.
2 https://drive.google.com/file/d/0BxJV2FFLpn5hXbXIVdEFFWmQ4U2M/edit?usp=sharing.
the project from the date of the first commit recorded in the project's VCS until the date of the last commit before the reporting date of the hundredth fixed bug report. Similarly, the late training set contains the data extracted from the source code of the project from the start of the project until the date of the first report in the Late test set.

Evaluating a bug assignment approach requires knowing the correct developers that fixed the bugs in the test set. To determine the correct developer for each test bug report, the source code-based technique proposed in our previous research (Shokripour et al., 2013) was used.

The source code of a project provides a rich information resource for recommending the most appropriate developer for fixing a new bug report. Recall that defining the identifiers is one of the activities of the developers when implementing the source code of the project. Therefore, the identifiers can be used to determine the type of the activities of the developers, or developers' expertise (see Section 3.1).

The method of extracting and decomposing the identifiers in this work is identical to the method used in our previous work (Shokripour et al., 2013). In this method, the identifiers and associated commit dates are extracted from the project's VCS. Then, the identifiers are decomposed into their components. The identifiers and the decomposed identifier components are used as the textual data in evaluating the approaches.

4.1.3. Baseline approaches for comparison

To determine the effect of using time-metadata in term weighting on the performance of an automatic bug assignment approach, the performance of the proposed approach (ABA-Time-tf-idf) was compared to the most commonly used ML and IR approaches in the literature. Cavalcanti et al. (2014) showed that SVM and Naive Bayes are the most commonly used ML methods and VSM is the most commonly used IR method in automatic bug assignment research. SUM was chosen due to its high performance in the mining software repository (Rao and Kap, 2011; Zhou et al., 2012) and automatic bug report assignment (Zhang and Lee, 2013) areas. A brief introduction of SVM, Naive Bayes, SUM, VSM follows.

- **Support vector machines (SVM):** SVM is a supervised learning algorithm that predicts the class for each new instance by determining linear or non-linear models that describe each class (Gunn, 1998). The SVM algorithm used in this work is a polynomial-kernel SVM which determines the maximum margin hyper-planes that classify the instances in the multidimensional space. The decision function, which is used to classify a new instance, is fully specified by a subset of training samples, called the support vectors. The attribute values of the instance are plugged into the obtained decision functions for the classes. The top N classes in the descending ranked list of the classes are predicted as a proper class for the new instance.

- **Naive Bayes (NB):** NB is a probabilistic technique that determines the probability of assigning an instance to a specific class using the Bayes conditional probability rule. In this technique, if a word in a class is more frequently used than in other classes, it increases the probability for a new instance that contains the word to belong to that class. In NB, the probability of each instance is determined by multiplying the probability of its features, and the probability of each feature is calculated based on its frequency in the class. This is due to the fact that if the frequency of one of the features is zero in the class, the probability of the class will be zero (Witten and Bell, 2006). This problem is commonly corrected using a smoothing method such as a Laplace estimator (Witten and Frank, 2000).

- **Vector space model (VSM):** VSM analyzes text documents in a training set and measures their similarity with a query in three steps. First, a vector of the terms in each of the documents is created. Next, the weights of the terms are determined for each document. Different term-weighting techniques have been used, with tf-idf being the most common (Salton et al., 1975). Finally, the cosine similarity of document vectors against a vector for the query is calculated and the documents are indexed based on their similarity with the query.

- **Smooth unigram model (SUM):** The unigram model (UM) is a simple case of a language model where the probability of each term is determined independent of other terms. The UM uses the term count frequencies in each document for calculating the probabilities. The problem of UM assigning a zero probability to the class that is missing a query term is resolved in the smooth unigram model (SUM) by measuring the probability of a term in all documents instead of a single document (Bai et al., 2004). In this paper SUM is used with the Dirichlet smoothing method (Eq. (6)).

\[
P(w | c_i) = \frac{c(w, c_i) + \mu P(w | C)}{|c_i| + \mu}
\]

(6)

The probability of the word \(w\) in class \(c_i\) is calculated using the count of the \(w\) in \(c_i\) \((c(w, c_i))\), the size of \(c_i\) \(|c_i|\), a pseudo-count \(\mu\) and the probability of the \(w\) in collection \(P(w | C)\).

For evaluating the SVM and Naive Bayes approaches, an implementation of the approach by (Anvik and Murphy, 2011) was used. For SUM, the implemented SUM in the Lingpipe3 library was used. For VSM, the implemented VSM method in the TraceLab4 framework was used (Dit et al., 2012).

4.2. Hypotheses

In this research, we had two main hypotheses with regard to the evaluation of the proposed term-weighting technique and the proposed automatic bug assignment approach. Accordingly, the null hypotheses are as follows:

1. \(H_0_{\text{ABA-Time-tf-idf}}\): There is no significant difference between the use of tf-idf and Time-tf-idf.
2. \(H_0_{\text{ABA-Time-tf-idf, Baselines}}\): There is no significant difference between the results of the ABA-Time-tf-idf approach and baseline approaches for automatic bug assignment.

The second hypothesis is expanded for all of the baseline approaches (SVM, NB, VSM, SUM). The derived hypotheses are as follows:

1. \(H_0_{\text{ABA-Time-tf-idf, SVM}}\): There is no significant difference between the results of the ABA-Time-tf-idf approach and SVM.
2. \(H_0_{\text{ABA-Time-tf-idf, NB}}\): There is no significant difference between the results of the ABA-Time-tf-idf approach and NB.
3. \(H_0_{\text{ABA-Time-tf-idf, VSM}}\): There is no significant difference between the results of the ABA-Time-tf-idf approach and VSM.
4. \(H_0_{\text{ABA-Time-tf-idf, SUM}}\): There is no significant difference between the results of the ABA-Time-tf-idf approach and SUM.

If a null hypothesis can be rejected with high confidence (95%), the corresponding alternative hypothesis, which is the inverse of the null hypothesis, can be supported.

4.3. Experimental design

This section describes the evaluation of the new proposed approach and the baseline approaches. The evaluation is done using both descriptive and statistical analyses. The metrics and methods used to do these analyses are detailed in this section.

---

4.3.1. Descriptive analysis (metrics)

To perform a descriptive analysis of the experiments, a set of metrics was identified to evaluate the results. The metrics used are as follows:

- **Top N rank or accuracy** (Rao and Kak, 2011; Zhou et al., 2012): For a new bug, if the top N ranked results contain the actual developer who fixes the bug, it is counted as a correct answer. In this case, the higher the value of the metric, the better the accuracy of the approach.

- **Effectiveness** (Dit et al., 2013; Liu et al., 2007; Poshvyvanyk et al., 2007): In bug assignment, effectiveness is defined as the position of the first relevant developer in the ranking. Those approaches that rank relevant developers at the top of the list are deemed more effective because they reduce the number of false positives that a developer has to consider. For this metric, the lower the value, the less effort is required by the triager, leading to a more effective approach.

- **Mean reciprocal rank (MRR)** (Baeza-Yates and Ribeiro-Neto, 1999): The reciprocal rank is the inverse of the rank position of the first relevant developer. In fact, it is the inverse of the effectiveness metric. The MRR is the average of the reciprocal ranks of a set of queries and is calculated using Eq. (7). In this equation, \( Q \) is the number of queries in the test set (i.e. the number of bug reports used for testing). The higher the MRR value, the better the performance of the approach.

\[
MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{Rank}_i}
\]

(7)

4.3.2. Statistical analysis

In this research, a statistical analysis is also used to evaluate the null hypothesis. In this case, the results from the approaches are investigated to determine the significance of the differences between them. From the identified metrics, the results of the effectiveness metric can be statistically analyzed due to the high number of output results (more than 60 output results). Due to the non-normality of the results for the effectiveness metric, a non-parametric Wilcoxon Matched-Pairs Signed-Ranks test was conducted. The significance level of the tests is \( \alpha = 0.05 \), meaning that if the \( p \)-value is lower than 0.05, the difference is statistically significant.

For the accuracy and MRR metrics there are very few data points; five for accuracy and one for MRR. Although a statistical test could be performed for these metrics, the results would not be meaningful due to the small sample size. Therefore, statistical tests were not performed for accuracy and MRR metrics.

4.4. Data collection and preparation

The data for evaluating the proposed approach was collected from the ITS (issue tracking system) and VCS (version control system) of the subject systems. Following the proposed steps in Section 3.1 and using text mining tools, the required terms were extracted from the source code files. From the ITS, the fixed bug reports were collected in XML format. Next, the fixed bugs were linked to the associated developers using two techniques: first, by examining any patch(es) attached to the bug report to extract the name of the developer, and, second, by mining the commit messages to find the bug’s ID.

The detection of bug IDs in commit messages was done using general patterns observed to be used by developers. Specifically, a rule-based named entity recognition (NER) method was used by applying the NE transducer component of the ANNIE\(^{5}\) plugin of GATE.\(^{6}\)

To improve the accuracy of the results from the NER, the extracted value from the commit messages was compared with the list of bug IDs collected from the ITS. Next, the commit date was compared to the date of creation and resolution of the bug report. If the date of the commit was after the creation date of the bug or before the resolution date of the bug, then the developer IDs from the commit and bug report were compared. If there was a match between these two, the commit was linked to the bug report and the committer was taken as the fixer of the bug. As individuals may have different user names in both the VCS and the ITS, a map was created by hand to associate user names from the two systems.

After linking the bugs to their developer, the summary and description fields of the fixed bugs were inputted into the ANNIE plugin to extract the nouns. The nouns were further refined by removing those that were less than three characters and symbols. The remaining nouns were then lemmatized using the Stanford CoreNLP API.\(^{7}\)

The collection of the lemmatized nouns was used as the dataset in the evaluation process.

4.5. Experimental implementation

The proposed approach was implemented in the TraceLab framework\(^{8}\) as a component. TraceLab is a framework to evaluate traceability methods. The collected data from the software repositories was preprocessed and the extracted terms with their associated metadata were imported into the implemented component. In our Tracelab experiment (Fig. 2), the Queries importer and Corpus importer components provided by Moritz and his colleagues (Dit et al., 2012) were used for importing the queries and corpus. The output of this experiment is a ranked list of developers in descending order. In this experiment, the CSV Similarity Matrix Exporter component of TraceLab was used for obtaining the list of the recommended developers in a comma-separated values (CSV) file (Fig. 2). The experiment is available online\(^{9}\) for reproducibility of this work by other researchers.

5. Evaluation results and analysis

To assess the proposed approach, an experimental evaluation was conducted. The results of this evaluation were analyzed to find answers to the two research questions: (1) Does using the time-metadata in determining the weight of terms improve the accuracy of an automatic bug assignment approach? and (2) Does the lightweight automatic bug assignment approach perform better than the most commonly used ML and IR automatic bug assignment approaches?

In these experiments, the accuracy, effectiveness and MRR metrics for the approaches were determined. Moreover, as mentioned in Section 4.1.2, five random test sets and two test sets belonging to specific time periods of the subject systems were used. To avoid a monotonous discussion about the performance of the approaches for each of the top 1 to top 5\(^{10}\) results, the performance of the approaches in each of the random test sets is discussed using the average of the accuracies (average accuracy) (Table 3). Moreover, the mean of the average accuracies for each approach from five random test sets (mean average accuracy) is used to compare the overall accuracy of the approaches for all of the random test sets. Also, the mean of the average effectiveness (mean average effectiveness) for each approach on the five random test sets is used for comparing the overall effectiveness of the approaches.

---

5 http://www.aktors.org/technologies/annie/.
6 http://gate.ac.uk/.
10 Top 5 refers to the five developers who have obtained the highest expertise values relevant to the new bug. If the actual fixer of the new bug is in the top 5 recommendations, the recommendation is considered correct.
As mentioned in previous sections, both the ABA-Time-tf-idf approach and the ABA-tf-idf approach are composed of a term weighting technique and a simple ranking method. The only difference between the ABA-Time-tf-idf approach and the ABA-tf-idf approach is using time in determining the weight of the terms. Therefore, comparing the results of these two approaches can answer the first research question to determine the impact of using time-metadata on the performance of an automated assignment approach.

Table 3 shows the average accuracy for the approaches when using the five random test sets. Comparing the average accuracies for the ABA-Time-tf-idf approach and the ABA-tf-idf approach indicates that using time in determining the weight of the terms improves the accuracy between 10.4–13%, 1.8–4.8% and 9–11.6% on the Eclipse, NetBeans and ArgoUML projects, respectively. The superiority of the ABA-Time-tf-idf approach to the ABA-tf-idf approach is observed by comparing the mean average accuracy and mean MRR for the ABA-Time-tf-idf and ABA-tf-idf approaches (Fig. 3). The obtained values for the mean average accuracy, the mean average effectiveness and the mean MRR (Table 4) indicate an overall superiority of the ABA-Time-tf-idf approach over the ABA-tf-idf approach for accuracy, effectiveness and MRR when using the random test sets. Moreover, Table 5 shows the average accuracy of five random test sets when recommending the top 1 to top 5 developers. These results show that when recommending the top 5 developers, the ABA-Time-tf-idf approach improves the accuracy 14%, 4% and 10% on Eclipse, NetBeans and ArgoUML projects, respectively. In addition, comparing the accuracy between the ABA-Time-tf-idf and ABA-tf-idf approaches for the Early test set (Table 5) shows that the accuracy has improved for the ABA-Time-tf-idf approach between 1–5%, 1–3% and up to 8% on Eclipse, NetBeans and ArgoUML projects, respectively, when recommending the top 1 to top 5 developers (Fig. 5). On the other hand, the results for the Late test set (Table 5) show a more significant improvement between 9–25%, 5–18% and up to 24% on the Eclipse, NetBeans and ArgoUML projects, respectively for the ABA-Time-tf-idf approach (Figs. 5 and 6).

To answer the second research question, the ABA-Time-tf-idf approach is compared to the baseline automatic bug assignment approaches. The overall results for each subject system are presented in Table 5.

The average accuracies for the SUM, VSM, NB, SVM and ABA-Time-tf-idf approaches using the five random test sets indicate the high performance of the ABA-Time-tf-idf approach for recommending...
Fig. 3. Comparing the accuracy and MRR of the ABA-Time-tf-idf approach with the SVM, NB, VSM, SUM and ABA-TF-idf approaches based on random test sets.

Table 4

Comparing the ABA-Time-tf-idf approach with the SVM, NB, VSM, SUM and ABA-Tf-idf approaches on subject systems based on random test sets.

<table>
<thead>
<tr>
<th>Project</th>
<th>Metric</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SVM (%)</td>
<td>NB (%)</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Mean average accuracy</td>
<td>2.36</td>
</tr>
<tr>
<td></td>
<td>Mean average effectiveness</td>
<td>24.72</td>
</tr>
<tr>
<td></td>
<td>Mean MRR</td>
<td>6.16</td>
</tr>
<tr>
<td>NetBeans</td>
<td>Mean average accuracy</td>
<td>1.92</td>
</tr>
<tr>
<td></td>
<td>Mean average effectiveness</td>
<td>42.86</td>
</tr>
<tr>
<td></td>
<td>Mean MRR</td>
<td>3.41</td>
</tr>
<tr>
<td>ArgoUML</td>
<td>Mean average accuracy</td>
<td>6.16</td>
</tr>
<tr>
<td></td>
<td>Mean average effectiveness</td>
<td>22.34</td>
</tr>
<tr>
<td></td>
<td>Mean MRR</td>
<td>7</td>
</tr>
</tbody>
</table>

The bold values indicate the best value for the metric for the different approaches.

Fig. 4. Comparing the average effectiveness of the ABA-Time-tf-idf approach with the SVM, NB, VSM, SUM, ABA-Tf-idf approaches using the random test sets.

the most appropriate developers. Comparing the average accuracies for these approaches using the five random test sets of the Eclipse project shows that the ABA-Time-tf-idf approach has improved the accuracy between 26–37.2%, 3.4–14.4%, 5.6–17.2% and 12.6–19.8% in comparison to the SVM, NB, VSM and SUM approaches, respectively. For the NetBeans project, the superiority of the ABA-Time-tf-idf approach over the SVM, NB, VSM and SUM approaches is between 41.2–52.2%, 23.2–32.2%, 5.6–14.8% and 1.8–4.8%, respectively. Finally, the ABA-Time-tf-idf approach has significantly improved the accuracy between 17.4–30.4%, 10–20%, 6.6–17% and 6–12.4% in comparison to the SVM, NB, VSM and SUM approaches for the five random test sets from the ArgoUML project. Despite selecting the data for the five random test sets from various periods of the subject systems' lifetime, the results show that the ABA-Time-tf-idf approach is the preferred approach for all subject systems (Fig. 3).

The mean average accuracies for the approaches on the subject systems also indicate a better performance of the ABA-Time-tf-idf approach over the ML and IR approaches. Table 4 presents the mean average effectiveness and mean MRR. These results show that the ABA-Time-tf-idf approach improved the MRR between 24.24–55.54%, 5.85–30.88% and 7.10–15.37% compared to the SVM, NB, VSM and SUM approaches. The lower effectiveness (Fig. 4) and higher MRR values for the ABA-Time-tf-idf approach compared to the other approaches (Fig. 3) indicates that ABA-Time-tf-idf ranks the actual fixer higher in the recommendation list.

Evaluating the approaches using the Early and Late test sets shows a different aspect of the performance of the approaches. Comparing the accuracy of the ABA-Time-tf-idf approach with that of the other approaches in the Early test set (Table 5) shows that the ABA-Time-tf-idf approach outperforms the SVM, NB, VSM and SUM approaches...
Table 5

<table>
<thead>
<tr>
<th>Project</th>
<th>Approach</th>
<th>Top 1</th>
<th>Top 2</th>
<th>Top 3</th>
<th>Top 4</th>
<th>Top 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg. randoms</td>
<td>Early</td>
<td>Late</td>
<td>Early</td>
<td>Late</td>
<td>Early</td>
</tr>
<tr>
<td>Eclipse SUM (%)</td>
<td>62%</td>
<td>22%</td>
<td>12%</td>
<td>4%</td>
<td>2%</td>
<td>0%</td>
</tr>
<tr>
<td>VSM (%)</td>
<td>62%</td>
<td>21%</td>
<td>7%</td>
<td>2%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>NB (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>SVM (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>ABA-tf-idf (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>ABA-Time-tf-idf (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>NetBeans SUM (%)</td>
<td>62%</td>
<td>21%</td>
<td>7%</td>
<td>2%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>VSM (%)</td>
<td>62%</td>
<td>21%</td>
<td>7%</td>
<td>2%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>NB (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>SVM (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>ABA-tf-idf (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>ABA-Time-tf-idf (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>ArgoUML SUM (%)</td>
<td>62%</td>
<td>21%</td>
<td>7%</td>
<td>2%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>VSM (%)</td>
<td>62%</td>
<td>21%</td>
<td>7%</td>
<td>2%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>NB (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>SVM (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>ABA-tf-idf (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>ABA-Time-tf-idf (%)</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
<td>7%</td>
<td>3%</td>
<td>0%</td>
</tr>
</tbody>
</table>

The bold values indicate the best accuracy for each approach.

In most of the cases (top 1 to top 5). For the Eclipse project, there are a few exceptions where VSM (e.g. top 4) and NB (e.g. top 1) are better than the ABA-Time-tfidf approach (Fig. 5). However, this 1% difference is not significant. The only notable exception to ABA-Time-tfidf’s superiority over the other approaches is for the Early test set where there are higher accuracies for the Naive Bayes approach in four cases (top 1, top 2, top 3 and top 4) on the ArgoUML project (Table 5). However, when using the Late test sets, the ABA-Time-tfidf approach performs better than all the other approaches. The only exceptional case is the top 1 of the ArgoUML project where the accuracy of the SUM approach is 2% better than the ABA-Time-tfidf approach (Fig. 6).

5.1. Statistical results

Statistical tests were conducted to analyze the significance of the differences between the effectiveness results obtained from the ABA-Time-tfidf approach and the ML and IR approaches. The results of each random test set were analyzed separately. First, the normality of the results was evaluated. The results of both the Kolmogorov-Smirnov and Shapiro–Wilk normality tests indicate that the p-value is less than 0.05 (p-value < α). This indicates a non-normal distribution of the results. Furthermore, the Skewness and Kurtosis values were between 2 and -2, confirming the results of the Kolmogorov–Smirnov and Shapiro–Wilk tests.

Accordingly, a non-parametric Wilcoxon Matched-Pairs Signed-Ranks test was conducted to evaluate the significance of the differences. Table 6 presents the p-value of the comparative statistical test. As shown in this table, almost all of the p-values indicate a significant difference between the ABA-Time-tfidf approach and all other approaches. The one exception is for the Eclipse project where the results of the ABA-Time-tfidf approach are not significantly better than the NB approach on two of the five random test sets.

5.2. Discussion

In this section, the results from the descriptive and statistical analysis are used to answer the determined research questions of this research.

5.2.1. Impact of using time-metadata

The descriptive results show that the ABA-Time-tfidf approach is more accurate and more effective than the ABA-tfidf approach, meaning that Time-tfidf is more effective than tf-idf. The statistical analysis on the effectiveness results indicates the rejection of \( H_0: \text{Time-tfidf} \neq \text{tf-idf} \) and the acceptance of the alternative hypotheses. These results show that the consideration of time significantly improves automatic bug assignment in terms of accuracy, effectiveness and MRR.

Accordingly, we can answer the first research question. As previously mentioned, an important difference between ABA-tfidf and ABA-Time-tfidf is how term weighting is done. The ABA-tfidf approach uses the tf-idf term weighting technique and ABA-Time-tfidf uses the Time-tfidf term weighting technique. All other aspects of these two approaches are the same. Thus, the results of this experiment can be used to compare the term weighting techniques. The descriptive metric results show that augmenting the term weighting technique with time-metadata makes significant improvements in bug report assignment, and the statistical results show the significance of these improvements. These results strongly support the fact that determining the weight of the terms using time-metadata has a significant impact on improving the accuracy, MRR and effectiveness of the automatic bug assignment system which answers RQ1.

Furthermore, consideration of time in term weighting can be used for determining the areas of developer activity at different points in time of a project’s lifetime. The proposed approach uses a time
The importance of the role of keywords is not reduced. In other terms, a term in the context of the developer’s activities, the term will be frequently used by a developer, thus showing the importance of the term. However, if a term was used a long time ago, it avoids the need to determine an activity threshold for developers on a feature related to a new bug report. This consideration of time frequency. It also accounts for the time that a developer last worked on a feature, based on their importance in the project, as indicated by the term. A time-based term-weighting technique to adjust the weight of the terms can result in different results for the ML and IR approaches. Different aspects of the projects, such as size or development process, may not be possible to recommend one approach as the best automatic bug assignment approach for all projects (i.e., one-size-fits-all). These results indicate that it may not be possible to recommend one approach as the best automatic bug assignment approach for all projects (i.e., one-size-fits-all). Different aspects of the projects, such as size or development process, can result in different results for the ML and IR approaches.

5.2.2. Performance of other approaches in various conditions
The analysis of the mean average accuracy results of the most commonly used ML and IR approaches (Table 4) shows that different approaches work best for different subject systems. For example, NB, VSM, and SUM were observed to be the best approaches for Eclipse, NetBeans, and ArgoUML, respectively. These results indicate that it is not possible to recommend one approach as the best automatic bug assignment approach for all projects (i.e., one-size-fits-all). Different aspects of the projects, such as size or development process, can result in different results for the ML and IR approaches.
Consideration of time-metadata in weighting the terms, such as in the ABA-Time-tf-idf approach, can cause the term weighting process to correspond better with reality. In other words, the value of each term can reflect the conditions of the projects in a specific period of the projects' lifetime. Therefore, changes in the information resources of the project have less of an impact on the ABA-Time-tf-idf approach than for other ML and IR approaches. Evaluating and comparing the ABA-Time-tf-idf approach with different ML and IR approaches, using test sets from specific periods of the projects’ lifetime, indicates the efficiency of the ABA-Time-tf-idf approach.

The robustness of ABA-Time-tf-idf was shown by the superiority of the approach for most of the cases across all of the subject systems and datasets. The statistical analysis shows a significance of the differences in most of the cases. The only exception was for H0:ABA-Time-tf-idf\_NB vs NB where for two out of five random test sets, the p-values did not indicate statistical significance. However, the average effectiveness in all cases showed an improvement, as well as a notable improvement in accuracy.

The considerable differences between results for the Early and Late test sets shows the impact of the use of time-metadata on the performance of the approach. The bug reports of the Early test sets belong to the start of the project, where there is little information in the projects' information resources and little time distance between the activities of the developers on the source code. Accordingly, time-metadata has less of an impact on the performance of the approach. Conversely, in the late period there is more information in the information resources of the projects. Therefore, time-metadata has a significant impact in distinguishing developers’ activities and determining the value of each activity at that time. Nevertheless, due to the increasing number of project developers and incremental changes in where developers work in the system during software development, identifying the correct developer for bug fixing becomes more challenging, time-consuming and tedious. Thus, as the project evolves over time, the need for a more accurate automatic bug assignment approach becomes more important. Therefore, having an approach with high accuracy in the middle and late period of the project development is more necessary than in the early period of project.

To summarize the analysis of the results, and answer the second research question, we can state that using Time-tf-idf in an automatic bug assignment approach and ABA-Time-tf-idf outperforms the most commonly used ML and IR automatic bug assignment approaches in most of the cases.

6. Threats to validity

In this section, we present threats to the validity of this work. Specifically, threats to the internal and external validity are discussed.

6.1. Internal validity

In this study, we assumed that the identifiers were appropriately named. To mitigate the risks of this assumption, we selected well-managed projects as the subject systems.

We used text mining methods to analyze the text resources. However, there exists some noisy text in the information resources such as stack traces in bug reports. This noise can cause the text mining methods to incorrectly determine the category of some terms. However, we found through inspection that the number of incorrectly determined categories was negligible.

Unlike normal text documents that can be assumed to follow specific grammatical rules, this assumption is not valid for the textual information resources of software projects. This can lead to textual analysis tools to incorrectly analyze some parts of the textual data. To overcome this problem, nouns extracted by the text analysis method were further refined.

This work, like other bug assignment approaches, assumes that there is only one developer that is working on a bug. However, some bugs require the collaboration of multiple developers to be fixed. This collaboration is often not reflected in the specific fields of the artifacts, such as an assigned-to field or the user name in a commit comment, which can contain only a single value. It cannot be said with certainty that all of the bug reports used in this work were only worked on by a single developer. However, examination of comments from a random set of bug reports for evidence of collaboration did not show evidence to the contrary. Also, the recommendation list could be considered the candidate list for cases where there may be collaboration.

In some projects, a few developers are the 'gatekeepers' for the project and only they have the necessary permissions to commit to the source repository. It is possible that for some of the software projects used in this work, the people who changed the source code are different from the people who committed the change(s) into the source repository. In this research we assumed that the developer who has committed the changes to repository is the actual fixer of the bug report.

Bug reports are created by the users of the software product. However, this group rarely has knowledge of the project's technical details. This can lead to the reporters using terms to describe the bugs that are different than the terms used by that project's developers. Although this issue may have had an impact on the results of the proposed approach, the subject systems were all developer tools. Therefore, the reporters can be expected to use a highly technical vocabulary that would be similar to that used by the project developers.

6.2. External validity

All of the datasets used in the experimental evaluation were selected from open-source projects that are written in the Java programming language and for which ITS-VCS links could be determined. The nature of the data in open-source projects may be different from proprietary projects. The projects were selected regardless of the quality of their data and bug reports in order to create unbiased test sets. It cannot be claimed that the results presented would extend to all other open-source or commercial projects. Therefore, more projects need to be studied to further validate the results in this paper.

As there is no accepted standardized test set for evaluating bug assignment, this remains a difficult issue. Specifically, the use of more data does not necessarily correspond to a rigorous evaluation. For example, increasing the size of the dataset also increases the probability of introducing noisy data that may lead to positively or negatively biased results (Kagdi et al., 2012; Linares-Vasquez et al., 2012). Furthermore, the analysis of huge datasets can cause challenges like impractically long runtimes. Accordingly, three subsets of fixed bugs were selected as random, Early and Late test sets. The sizes of the test sets used in this evaluation are not uncommon (Anvik and Murphy, 2011; Linares-Vasquez et al., 2012). Our test set size is not as high as that used in (Bhattacharya et al., 2012; Tamravi et al., 2011) nor is it as low as the one used in (Kagdi et al., 2012; Zhang and Lee, 2013). Nevertheless, this topic remains as a part of our future work.

7. Related work

In automatic bug assignment approaches, researchers have used various ML and IR methods to analyze the textual information resources of the software repositories. Cavalcanti et al. (2014) showed that ML and IR methods have been widely used by researchers for improving automatic bug assignment systems. Therefore, this section considers the recommended approaches which use the most commonly used ML and IR approaches for automatic bug assignment. Moreover, Cavalcanti et al. found that most of the approaches used tf-idf for determining the weight of the terms.
The approaches that used ML methods either used these methods alone or in conjunction with other techniques and metadata. Cubranic and Murphy (2004) recommended an automatic bug assignment system based on machine learning techniques. They used a Naïve Bayes classifier for determining the similarity between the expertise of the project’s developers and the new bug report. Anvik and Murphy (2011) considered the performance of various ML methods on automatic bug assignment. They found that SVM had the best performance when compared with other ML methods. Ahsan et al. (2009) tried to improve an automatic bug assignment system by applying dimension reduction techniques, such as feature extraction (LSI) and feature selection, on the extracted data. They used machine learning methods to recommend the most appropriate developers based on the reduced dataset. Their evaluation demonstrated that SVM outperforms other ML methods they considered. Nasim et al. (2011) further recommended a new automatic bug assignment which used a machine learning algorithm method for classifying the bug reports based on their fixers using the frequency of the alphabets in the bug reports. They found that Simple logistic, SMO and Complement Naïve Bayes algorithms performed better in comparison with the other investigated ML algorithms.

Jeong et al. (2009) recommended using tossing graphs for improving automatic bug assignment. The bug tossing graph is generated based on the bugs reassigned by the developers. In this method, the tossing information is used in addition to ML methods for improving automatic bug assignment approach. They considered the performance of their method using two machine learning algorithms, Naïve Bayes and Bayesian Networks, on data from the Eclipse and Mozilla projects. Bhattacharya et al. (2012) used an improved tossing graph in addition to ML methods in their approach. They labeled the edges of the tossing graph using the expertise of the developer and the nodes of the tossing graph with the activity of the developer, which significantly decreased the tossing graph path lengths. Their evaluation showed that Naïve Bayes had the best performance on their subject systems.

Xia et al. (2013) recommended a composite semi-automatic method that combined bug report analysis and developer analysis for recommending developers. They used multi-label k-nearest neighbors (ML-KNN), a ML-based classifier, to determine the similarity of previously fixed bugs to new bug reports and the expertise of the developers. They used four features of each bug: term, topic, component, and product, to determine the similarity.

Banitaan and Alenezi (2013) recommended a new method that used the X2 method for dimension reduction, in addition to a ML method, to improve automatic bug assignment. To construct the vector space model, they used 1% of the most discriminating terms of the bug descriptions as well as the component and reporter features of the bug as metadata.

Nguyen et al. (2012) proposed a topic-based automatic bug assignment method. In this method, the bugs of the training set are classified based on their topics. A latent Dirichlet allocation (LDA)-based automatic topic analysis was used for classifying the bugs based on their descriptions. They used the time to resolve the bugs of each topic as metadata for recommending the developers with the most expertise for each topic.

Wu et al. (2011) suggested a ML-based approach that used the K-nearest-neighbor algorithm to classify the bug reports. Their method included two components. The first component used the VSM method with tf-idf term weighting for converting the fixed bug reports into a vector space and determining similar bug reports to a new bug report. Then, the second component used frequency and social network metrics for ranking the developers based on participation records of developers in discussing similar bug reports.

Furthermore, various IR methods have been used in proposed approaches for improving the performance of automatic bug assignment systems. As mentioned in Section 4.1.3, the VSM method is one of the most used IR methods for determining the similarity between a new bug report and the expertise of developers (Baysal et al., 2009; Chen et al., 2011; Matter et al., 2009). Moreover, some other IR methods, such as latent semantic indexing (LSI) as an extension of classical VSM (Kumar et al., 2012), have been used by researchers (Kagdi et al., 2012; Linares-Vasquez et al., 2012). The research of Zhang and Lee (2013) recommended a new method based on the combination of an experience model and a probability model. First, fixed bug reports which are similar to a new bug report are extracted using the smooth unigram model (SUM) method. Then a probability model and an experience model are created based on the similar bug reports. For creating the probability model, social network techniques are used to determine the relationship between the developers from the comments in the bug reports. Then the experience model is created based on some activity factors of the developer for the project (e.g. number of fixed bug reports by the developer). Finally, these two models are combined together to determine the rank score of the developer for the new bug report.

8. Conclusion

This paper proposes a new term-weighting technique for improving automatic bug assignment methods based on the tf-idf term-weighting technique. Unlike the traditional tf-idf technique that only deals with the frequency of use for weighting the terms, the proposed technique, Time-tf-idf, also considers the time when the term was used in the resources. An evaluation of the ABA-Time-tf-idf approach shows an improvement in the accuracy and MRR of the approach by up to 14% and 9%, respectively, in comparison with the ABA-tf-idf approach. Moreover, the ABA-Time-tf-idf approach had an improved accuracy over the SVM, NB, VSM and SUM approaches up to 45%, 28%, 11% and 19%, and an improved MRR by as much as 55.54%, 30.88%, 15.37% and 25.03%, respectively, for three subject systems. In addition, the results show that while the most commonly used ML and IR methods had different performances on the various subject systems, the ABA-Time-tf-idf approach was superior for all of the subject systems. This experimental evaluation shows that a combination of the Time-tf-idf term weighting technique and simple ranking (ABA-Time-tf-idf approach) is not only lightweight with respect to computations in determining the expertise of developers, but also obtains a high accuracy and MRR when compared with other ML and IR approaches for automatic bug assignment. Furthermore, considering time-metadata in bug assignment gives more weight to later developer activities that removes the need for using inflexible developer activity thresholds.

Facilitating and improving the bug assignment process will improve the overall quality of the software project by reducing the time and cost for software evolution. With regards to the results, Time-tf-idf is a robust term weighting technique for enhancing the bug assignment process and consequently, it is worthy for inclusion in software development processes.

In future work we plan to compare the ABA-Time-tf-idf approach with state of the art approaches such as association and topic-based approaches (Tamravi et al., 2011; Xia et al., 2013).

References


Sima Zamani

Sima Zamani is an assistant professor at the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. Her research interests are in the area of mining software repository and software maintenance.

John Anvik

John Anvik is an assistant professor at Central Washington University, USA, where he trains future generations of software developers. He started researching ways to improve bug report triage while at the University of British Columbia, Canada.

Zarinah Mohd Kasirun

Zarinah Mohd Kasirun received her BSc (CS) and MSc (CS) from National University of Malaysia (UKM) in 1989 and 1993, respectively. On August 2009, she received her doctoral degree from University of Malaya. Currently she is an Associate Professor in Software Engineering Department at the Faculty of Computer Science and Information Technology, University of Malaya. She has 13 years of experience in teaching at undergraduate and postgraduate levels. Published many academic papers in conferences and journals. She actively supervises many students at all levels of study – Bachelor, Master and PhD. Her interest of research includes requirements engineering, requirements visualizations, software metrics and quality, object-oriented software engineering, computer-supported collaborative learning.

Ramin Shokripour

Ramin Shokripour is a PhD candidate in the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. His research interests are in the area of mining software repository and software maintenance.