Learning to Classify Bug Reports into Components

Ashish Sureka

Indraprastha Institute of Information Technology (IIIT-D)
New Delhi, India
ashish@iiitd.ac.in

Abstract. Bug reports in widely used defect tracking systems contains standard and mandatory fields like product name, component name, version number and operating system. Such fields provide important information required by developers during bug fixing. Previous research shows that bug reporters often assign incorrect values for such fields which cause problems and delays in bug fixing. We conduct an empirical study on the issue of incorrect component assignments or component reassignments in bug reports. We perform a case study on open-source Eclipse and Mozilla projects and report results on various aspects such as the percentage of reassignments, distribution across number of assignments until closure of a bug and time difference between creation and reassignment event. We perform a series of experiments using a machine learning framework for two prediction tasks: categorizing a given bug report into a pre-defined list of components and predicting whether a given bug report will be reassigned. Experimental results demonstrate correlation between terms present in bug reports (textual documents) and components which can be used as linguistic indicators for the task of component prediction. We study component reassignment graphs and reassignment probabilities and investigate their usefulness for the task of component reassignment prediction.

Keywords: Mining Software Repositories (MSR), Empirical Software Engineering and Measurements (ESEM), Automated Software Engineering (ASE).

1 Research Motivation and Aim

Quality of bug reports submitted to defect tracking systems is a topic that has attracted a lot of research attention recently. Previous studies reveals that the quality of information present in a bug report influences its resolution time and has impact on the productivity of the development team [1][2][3][10][12]. Bettenburg et al. and Zimmerman et al. investigate quality of bug reports and mention that bug reports are often poorly written and provide *inadequate* and *incorrect* information. This naturally results in delays during problem identification and bug fixing. They conduct a survey with Apache, Eclipse and Mozilla project developers and present several factors impacting the quality of bug reports [2][12].

The difference between inadequate and incorrect information in bug reports is import to this work. Not providing important information such as steps-to-reproduce and stack-traces is one kind of quality issue. This is an issue of missing or inadequate information. However, bug reports contain certain mandatory and standard fields like product name, component name, version number, platform and operating system. Prior studies reports that bug reporters often provide incorrect values for such mandatory and standard fields. This is a different kind of issue (pertaining to incorrect information rather than missing or inadequate information) which is of interest to the work presented in this paper. For example, accidently providing incorrect information such as an incorrect operating system is a typical problem in bug reporting [12]. Zimmerman at al. mention that several bug reports in Eclipse project were submitted for "Windows" but later reassigned to "Linux". Their study also reveals that for bug reporters locating a *component* in which a bug occurs is often difficult and impossible to provide [12].

The focus of this work is to investigate the phenomenon of incorrect information in standard fields and in particular study the phenomenon of *incorrect component assignment.* An interesting result of the survey conducted by Zimmerman et al. reveals that information like product name, component name, version number and operating system are important items used by developers during bug fixing and more importantly it causes them problems and delays in bug fixing if such information is wrongly provided [12].

Zimmermann et al. perform an experiment using Eclipse bug reports to construct a decision tree that correlates input features extracted from bug reports (bug severity, operating system, affected component, bug priority, version number, reporter name and platform) and the location of the fix. Their study reports three out of the seven input features influenced the location of a defect. They demonstrate that the input feature with the most impact in determining the defect location is *component name* (due to the direct mapping between source code and component name). They found that the next most influential feature is the version number [13]. Breu et al. analyze frequently asked questions by users and developers in bug reports [5]. They observe that often bugs are submitted to incorrect components or even projects. For example, some of the actual questions used in their paper are: "Is dom the correct component?", "Can you verify that this is a reconciler problem?" "Should I move this bug report to JCore-code assist?" Breu et al. mention that 5 out of the 94 questions (around 5-6%) they analyzed on bug triaging were pertaining to component correction [5]. The research performed by Breu et al. shows that bug reporters often mention incorrect components and developers ask questions to bug reporters for clarification resulting in delays. The research presented in this work is motivated by the following facts.

1. Mandatory fields like *component name* are important indicators for determining the location of a defect in the source code.
2. Bug reporters often provide *incorrect* information for standard fields like *component name.*

3. Wrong information for fields like component name can cause delays and problems to developers in bug fixing.

The specific *research aim* of the work presented in this paper is the following:

1. To perform empirical analysis and measurements on component reassignments phenomenon by mining data archived in defect tracking systems.
2. To investigate the usefulness of machine learning based techniques (text classifiers) and framework for the task of automatically classifying the correct component for bug reports.
3. To investigate the usefulness of machine learning based techniques for the task of predicting if a component reassignment event will occur or not given the initial component assigned by the bug reporter.
4. To study component reassignment graphs and component reassignment probability and investigate their usefulness in improving the accuracy of correct component prediction.

2 Related Work and Research Contributions

Previous research shows that identifying the most competent developer to fix a given bug report is a challenging task and is error prone [4][7][8][11]. The work done on the topic of bug tossing (reassignment of bug reports to various developers) is most closely related to the research study presented in this paper. Recently several researchers have looked into the phenomenon of bug tossing and proposed several techniques (based on machine learning and tossing graphs) to perform automatic triaging and reduce the percentage of bug tossing events [4][7][8][11]. We briefly review closely related work and list some of the key differences between prior study and the work presented in this paper.

To the best of our knowledge and literature study, the initial work on the top of bug tossing was performed by Jeong et al [11]. Jeong et al. introduced the idea of deriving bug tossing graphs from historical data and employ a graph model based on Markov chains to better assign developers to bug reports. Their study reveals that tossing increases the time-to-correction for a bug and show that a significant percentage (about 40%) of bugs (Eclipse and Mozilla project) have been tossed (reassigned to a different developer) at-least once [11]. Bhattacharya et al propose a method (an extension to prior triaging approaches) to reduce tossing path lengths and validate their approach on popular open-source projects like Eclipse and Mozilla [4]. Chen et al. presents an approach that computes textual similarity between bug reports using a vector space model and uses this information in conjunction with tossing graphs to improve bug assignments [7]. Guo et al. mention that bug report reassignments is a commonly occurring phenomenon during the bug fixing process and yet has rarely been studies. They present a large-scale quantitative and qualitative analysis (in Microsoft Windows Vista operating system project) on the topic of reasons for the software bug report reassignments [8].

The key difference between previous work and this study is that while previous work focuses on *developer reassignment* or tossing, our research focuses on *component reassignment.* While there has been work done in the area of *bug report quality* [1][2][3][10][12] and *bug tossing* (reassignment of a bug report to developers) [4][9][11], we identify a *research gap* and a phenomenon which is not yet explored in-depth in the literature on Mining Software Repositories. The problem of incorrect assignment of component field (crucial information for bug fixing) in a bug report is not yet studied closely and the objective of our research is to improve our understanding and throw light on this relatively unexplored phenomenon. In context to existing work, this paper makes the following *unique contributions*:

1. This paper is the *first* focused empirical study on the topic of *incorrect assignment* of *component* in bug reports. We perform experiments on bug report data downloaded from Eclipse and Mozilla (popular open-source projects) and report measurement results.
2. We perform a series of experiments using a *machine learning framework* for the task of predicting the correct component for a given bug report and report classification accuracy results.
3. We perform a series of experiments using a machine learning framework for the task of predicting if a component reassignment event will occur or not given the initial component assigned by the bug reporter.
4. We study *component reassignment graphs* and *component reassignment probabilities* and present our insights.

3 Empirical Analysis

3.1 Experimental Dataset

We perform empirical analysis on publicly available dataset from two popular open-source projects (Eclipse and Mozilla) so that our results can be replicated and used for benchmarking or comparisons. Table 1 presents the details of the experimental dataset obtained by downloading and parsing (XML and HTML files) bug reports from Bugzilla (the issue tracking system used by Eclipse[1] and Mozilla[2] Project) defect tracking systems. The experimental dataset consists of 28618 fixed bug reports from Eclipse project and 16268 bug reports from Mozilla project. Table 1 shows the number of distinct products and components for Eclipse and Mozilla projects. For example, JDT (Java development tools) and Platform (Eclipse platform) are products within Eclipse and APT and Core are components within the JDT product. Table 1 displays the number of distinct reporter, versions and assigned-to in the evaluation dataset. For example, the number of distinct products and components for the Eclipse dataset (28618 fixed bug reports) are 98 and 507 respectively.

[1] `https://bugs.eclipse.org/bugs/`

[2] `https://bugzilla.mozilla.org/`

Table 1. Experimental dataset details (open-source Eclipse and Mozilla projects)

	Eclipse	**Mozilla**
From Bug ID	200000	400000
To Bug ID	250000	450000
From Date (Reported)	Aug/2007	Oct/2007
To Date (Reported)	Oct/2008	Aug/2008
Fixed Bugs	28618	16268
Distinct Component	507	449
Distinct Product	98	44
Distinct Assigned To	1078	789
Distinct Version	140	70
Distinct Reporter	3439	2229

3.2 Statistics on Reassignment Events

Bug reports consists of fields like component, product, assignee and version which can be re-assigned after reporting or initial assignment. Guo et al. present a large-scale quantitative and qualitative analysis of the bug reassignment process in the Microsoft Windows Vista operating system project and categorized five primary reasons for reassignments of assignee fields: finding the root cause, determining ownership, poor bug report quality, hard to determine proper fix, and workload balancing [8]. We perform statistical analysis on the evaluation dataset to compute percentage of reassignment on four fields: component, product, assignee and version.

Tables 2 and 3 displays data on the percentage of fixed bug reports in the experimental dataset that have undergone component, product, assignee and version reassignments (the values are computed by extracting relevant information from the bug history available in Bugzilla defecting tracking system). The data in Tables 2 and 3 is presented in a format that enables comparison and contrast between percentage of reassignment across four fields (component, product, assignee and version). Empirical results in Table 2 reveals that the percentage of bug reports in Eclipse undergoing component reassignment is 15.75% (average of 98 products) and 39.63% for the EMF product (one of the products within Eclipse). As shown in Table 3, the percentage of bug reports in Mozilla project (average of 44 products) undergoing component reassignment is 24.21% which is much higher than the corresponding value in Eclipse. Based on the experimental results, we draw a conclusion that there is a significant percentage (15.75% for Eclipse project, 24.21% for Mozilla project) of fixed bug reports for which a component reassignment activity is performed (and hence we believe an interesting topic for scientific investigation).

We performed a manual inspection of bug reports in the valuation dataset and noticed a phenomenon wherein component, product, assignee and version reassignment happens multiple times. We conducted empirical analysis to compute the number of times component, product, assignee and version reassignments

Table 2. [*Eclipse Project*] NUM: number of bug reports (BRs), PER: % of BRs in the experimental dataset, %CR: % of BRs having component reassignment event, %PR: % of BRs having product reassignment event, %AR: % of BRs having assignee reassignment event, %VR: % of BRs having version reassignment event.

Product	**NUM**	**PER**	**%CR**	**%PR**	**%AR**	**%VR**
BIRT	2879	10.06	22.17	0.35	**94.34**	1.36
Community	1168	4.08	4.63	1.29	26.45	0.60
EMF	857	2.99	**39.63**	1.64	25.56	**37.81**
Equinox	1771	6.18	13.88	4.69	61.16	2.66
JDT	1449	5.06	18.88	**5.32**	79.92	4.56
PDE	1242	4.33	8.94	4.51	85.67	2.02
Platform	3083	10.77	13.92	5.23	77.36	4.38
Web Tools	116	0.40	13.80	1.63	37.07	1.63
All [**98**]	28618	100	15.75	6.77	61.77	9.88

Table 3. [*Mozilla Project*] Table 2 defines labels NUM, PER, %CR, %PR, %AR and %VR

Product	**NUM**	**PER**	**%CR**	**%PR**	**%AR**	**%VR**
addons	768	4.72	15.76	3.52	64.98	10.29
Calendar	658	4.04	11.71	1.07	74.02	17.63
Core	3952	24.29	25.26	17.34	64.00	11.11
Firefox	1811	11.13	**40.87**	3.48	55.22	**28.00**
Localiz.	837	5.14	11.59	6.82	24.14	1.08
mozilla.org	2396	14.72	32.06	11.02	**80.22**	11.27
Toolkit	734	4.51	20.58	**63.22**	70.71	25.21
All [**44**]	16268	100	24.21	16.45	61.97	15.62

happens during the lifetime (from creation to close) of each fixed bug in our experimental dataset (refer to Table 4. The data in Table 4 (Left: Eclipse Project, Right: Mozilla Project) reveals the exact percentages of fixed bug reports on which the reassignment activities (enabling comparison and contrast across four fields: component, product, assignee and version) are performed more than once. Table 4 indicates that for Eclipse project, 47.69% of bug reports in the evaluation dataset had one assignee reassignment event and 14.08% of the bug reports had more than one assignee reassignment event. For Mozilla project, 5.79% of the bug reports in the evaluation dataset had more than one component reassignment event. Based on the empirical results displayed in Table 4, we draw a conclusion that multiple reassignment (during the life-cycle of a bug report) of component, product, assignee and version is a phenomenon which is present in open-source projects like Eclipse and Mozilla. An interesting result is that amongst the four fields, assignee field is most frequently reassigned and component field is next (in terms of the percentage of reassignment activities) after assignee field.

We perform a manual inspection of the developer comments (online discussion

Table 4. [*Left: Eclipse Project, Right: Mozilla Project*] Distribution of bug reports across number of times (0,1,2,3, ≥ 4) a component, product, assignee and version reassignment event takes place within the lifetime of a fixed bug report

	0	**1**	**2**	**3**	**≥4**
Component	84.25	13.25	1.96	0.36	0.18
Product	93.23	6.42	0.27	0.06	0.02
Assignee	38.23	47.69	9.51	3.10	1.47
Version	90.12	8.57	1.20	0.06	0.05

	0	**1**	**2**	**3**	**≥4**
Component	75.79	18.42	4.56	0.86	0.37
Product	83.55	14.53	1.64	0.21	0.07
Assignee	38.03	50.19	7.49	2.95	1.34
Version	84.38	14.16	1.11	0.28	0.07

forum in Bugzilla) to understand the reason for component reassignment in bug reports. Table 5 displays some of the developer comments (entered in threaded discussion) throwing light on component reassignment phenomenon. The comments in Table 5 shows that indentifying the correct component is non-trivial and developers often make mistake in component assignment. As shown Table 5, bug reporters and developers are sometimes not sure of the correct component, often assign bug reports to wrong component and later perform reassignments and discuss (through online forums in Bugzilla) amongst the team to identify the correct component for a given bug report.

We measure the difference between the creation timestamp of the bug report and the timestamp for component and product reassignment events. Table 6 shows the minimum, maximum, lower and upper quartile values for the time difference (in hours) for fixed bug reports in the Eclipse and Mozilla experimental dataset. The median value for the Eclipse project (component reassignment) is 340 and the median value for the Mozilla project is 157. The numbers clearly indicate that component reassignment (or correction) is not immediate and the time difference between reporting time and first reassignment or between subsequent reassignments (for example, between first reassignment and second reassignment) is significant. Hence we believe that automated techniques that can guide developers for the task of correct component assignment aimed at reducing the percentage of reassignment events and time between reassignments can be useful for practitioners involved in the bug fixing process. Figure 1 shows differences phases in the life-cycle of an Eclipse bug report (BUGID 202407). As shown in Figure, the bug is reported at timestamp $T1$ (by developer Alex) and undergoes a component (from Data Access to Connectivity), product (BIRT to Data Tools) and version (2.2.0 to unspecified) reassignment event at timestamp $T2$. At timestamp $T3$, the bug report undergoes assignee reassignment (from birt-dataaccess-inbox to ichan) and version reassignment (from unspecified to 1.5). Component reassignment again happens at timestamp $T4$ (Connectivity to Open Data Access) and finally the bug report is fixed at $T5$ (by developer Ichan) and closed at $T6$ (by developer Bfitzpat). Figure 1 (depicting the bug history and actions perform on Eclipse BUGID 202407) indicates that component, product, version and assignee at the time of bug reporting was not correct as a result of which it undergoes reassignments. The component reassignment happens twice and states at timestamp $T2$, $T3$ and $T4$ are result of reassignments.

Table 5. Illustrative examples of developer comments (on the issue of component reassignment) in Bugzilla threaded discussions

Eclipse		Mozilla	
BUGID	**Comment**	**BUGID**	**Comment**
200513	Not sure if this is us or SWT.	433562	Then this is an addons.mozilla.org bug, not a download manager bug. I've already moved the bug to the proper component.
200849	This is probably the wrong component. We're looking to add a section to the help at help.eclipse.org.	432653	It's a "Provider:GDATA" problem and no "Calendar View" one
200889	I believe this has been incorrectly assigned to the ATNA component	432131	Probably not a theme issue, but I have no idea where to look for the real cause. Might be widget, might be something else.
202384	Bug was categorized as JDT/APT - I don't think it has anything to do with either. I'll attempt to recategorize it.	431665	Definitely a TE bug. the component where this bug now lives.
202893	I notice that this is assigned to User Assistance. Paul, do you think that the bug is in UA code or in Platform UI code?	431451	This should go to Server ops.

3.3 Automatic Component Assignment Using Linguistic Features

One of the research objectives of the work presented in this paper is to derive a statistical and probabilistic model from historical data to predict the correct component for a given bug report. We frame the problem of automatic component assignment as a text classification problem. The input to the classifier is a bug report (title and description) and the output is one of the category (or Top-K categories) from a pre-defined finite number of discrete categories or classes (software components). Every bug report is assigned to only one component (no cross-cutting categorization) and hence the output categories (in the text classification framework) are both exhaustive and mutually exclusive. Every bug report is assigned to exactly one of the pre-defined class. We leverage LingPipe[3] which implements several types of text classifiers and conduct a series of classification experiments.

We perform experiments to investigate the extent to which two different classifiers (TFIDF and DLM) implemented in LingPipe can be used to automatically predict the correct component of a bug report based on the title and textual description [6] provided in the bug report. TFIDF (term-frequency (TF) and inverse document frequency (IDF)) classifier belongs to the class of discriminative classifiers whereas DLM (Dynamic Language Model) is a language model classifier. We frame the task of inducing a classifier as a supervised learning task

[3] `http://alias-i.com/lingpipe/`

Table 6. Minimum, maximum, lower and upper quartile values for the time difference (in hours) between the bug creation timestamp and the component (CM) or Product (PC) reassignment timestamp for fixed bug reports in the Eclipse (EC) and Mozilla (MZ) experimental dataset

	EC-CM	EC-PD	MZ-CM	MZ-PD
Max	30506	30506	27628	29961
3rd Quart	7069	13873	5421	6540
Median	340	5203	157	1721
1st Quart	14	50	4	17
Min	0	0	0	0

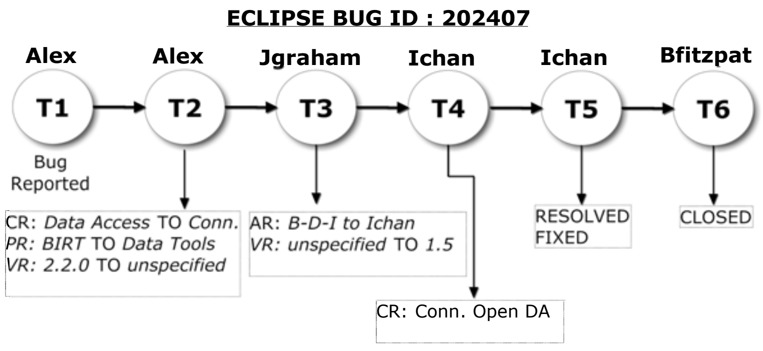

Fig. 1. Lifecycle of Eclipse BUGID 202407 showing PR (product reassignment), CR (component reassignment), AR (assignee reassignment) and VR (version reassignment)

(inferring a function or the classifier) which requires the experimental dataset to be divided into two sets: training (for learning a classifier) and test dataset (for evaluating the performance of the classifier). For TFIDF classifier, linguistic features from bug reports are extracted using the *IndoEuropeanTokenizerFactory* implemented in LingPipe. For the DLM classifier, we train a character based language models and set the size of the NGRAM to be 6 characters.

We divide the experimental dataset into training (bug reports from 20000 to 230000 for Eclipse and bug reports from 400000 to 430000 for Mozilla) and test dataset (bug reports from 23001 to 250000 for Eclipse and bug reports from 430001 to 450000 for Mozilla) and derived a predictive model for 12 products (number of components for each product is displayed in Tables 7 and 8). The ground-truth (correct component for each fixed bug report) is already available as all the bug reports in the evaluation dataset are fixed and closed. Tables 7 and 8 displays accuracy result for two classifiers (default settings of TF/IDF classifier and Dynamic Language Model classifier implemented in LingPipe text processing toolkit). We report Top N accuracy (considered a hit if the actual component is amongst the Top N component predictions) results and the performance results in Table 7 and 8 indicate that word-level and character-level features in

Table 7. Top N (N = 1,2,3,4 and 5) accuracy results (component prediction) for two machine learning (ML) classifiers (TFIDF and DLM) on *Eclipse* project dataset (COMP: Number of components, TRAIN: size of training dataset, TEST: size of test dataset)

PROD.	CMP	TRAIN	TEST	ML	TOP 1	TOP 2	TOP 3	TOP 4	TOP 5
Community	26	737	431	TFIDF	48.95%	65.88%	73.53%	79.09%	83.96%
				DLM	42.45%	56.37%	68.20%	74.69%	79.56%
Platform	17	2027	1056	TFIDF	46.68%	65.24%	76.50%	82.27%	85.86%
				DLM	54.07%	72.06%	78.49%	82.08%	84.44%
BIRT	12	1837	1042	TFIDF	56.33%	75.61%	83.95%	89.80%	93.63%
				DLM	62.18%	78.01%	85.78%	90.09%	93.06%
Equinox	9	1207	564	TFIDF	69.32%	87.05%	92.54%	96.08%	98.03%
				DLM	68.79%	84.92%	92.18%	95.90%	97.49%
JDT	6	969	480	TFIDF	60.62%	82.29%	94.16%	98.32%	99.80%
				DLM	62.29%	81.66%	92.91%	98.53%	99.58%
PDE	5	762	480	TFIDF	68.33%	85.41%	94.78%	98.11%	100%
				DLM	73.95%	86.03%	94.98%	97.06%	100%

free-form textual reports can be used as discriminatory signals for the task of automatic component classification. As shown in Table 7 (Eclipse project), the Top 1 accuracy for the DLM multi-class classifier varies from a minimum of 42.45% (26 mutually exclusive classes) to a maximum of 73.95% (5 mutually exclusive classes). Table 8 (Mozilla project) reveals that the Top 5 accuracy of the TFIDF classifier varies from a minimum of 63.86% (91 components) to a maximum of 91.88% (24 components). We perform empirical analysis to discover correlation or association between terms in bug reports and the components. The purpose is to understand and uncover the discriminatory terms used by the machine learning classifiers (such as the TFIDF classifier) to perform the text categorization task. We represent bug reports as bag of terms (generated by converting the sequence of characters into tokens using a tokenizer) and generate the distribution of words for each category or class. We compute the most frequent terms for each category and compute the probability of each term given a category. If the probability of a term in one specific category is high (i.e., the term is frequent or prevalent) and low in other categories then the term is a good discriminatory feature for the purpose of performing text classification.

We investigate the JDT product (Eclipse project) consisting of six components: APT, CORE, DEBUG, DOC, TEXT and UI. We compute the probability of each unique term in the training dataset corpus in each of the six categories. We observe that *java*, *eclipse*, *org*, *jdt* and *core* are frequent terms in all the six categories and hence are not discriminatory. However, we notice that there are certain terms which are discriminatory for a specific class. Table 9 reveals that the probability of the terms *type* or *junit* is 25 times more in class APT than in class DOC. The probability of the terms *jface* and *text* is significantly higher in TEXT component in contrast to other component. As indicated in Table 9, the probability of the term *kit* is very high in the DOC category in comparison to

Table 8. Top N (N = 1,2,3,4 and 5) accuracy results (component prediction) for two machine learning (ML) classifiers (TFIDF and DLM) on *Mozilla* project dataset (COMP: Number of components, TRAIN: size of training dataset, TEST: size of test dataset)

PROD.	CMP	TRAIN	TEST	ML	TOP 1	TOP 2	TOP 3	TOP 4	TOP 5
Core	91	2814	1138	TFIDF	38.57%	50.78%	56.84%	60.79%	63.86%
				DLM	36.11%	45.51%	50.95%	55.25%	57.79%
Toolkit	26	522	212	TFIDF	43.80%	62.37%	68.56%	75.70%	80.46%
				DLM	34.76%	50.47%	58.09%	64.75%	71.89%
Firefox	25	1415	396	TFIDF	50.75%	68.67%	77.50%	82.80%	85.57%
				DLM	53.53%	63.37%	70.94%	77.00%	80.53%
Mozilla.org	24	1321	1075	TFIDF	53.76%	75.89%	84.63%	89.28%	91.88%
				DLM	61.20%	80.45%	88.82%	91.14%	92.16%
Calendar	19	433	225	TFIDF	40.44%	59.55%	66.66%	76.43%	80.87%
				DLM	40.44%	56.44%	71.10%	79.54%	84.42%
addons.mozilla	16	495	273	TFIDF	55.67%	69.95%	80.20%	85.69%	90.45%
				DLM	58.97%	72.88%	82.77%	85.33%	87.52%

other categories. Terms *debug* and *lib* are common in bug reports belonging to the DEBUG component. The data in Table 9 clearly shows that certain terms are more common in bug reports belonging to one class in contrast to other classes. We perform similar experiments on Equinox product (consisting of nine components) of Eclipse platform (TFIDF classifier Top 1 accuracy is 69.32% and DLM classifier Top 1 accuracy is 68.79%) and observed correlation between certain terms and specific classes. We observe that terms *eclipse*, *org*, *equinox* and *java* are frequent in majority of the classes and hence do not have discriminatory power. We discover that terms *password* and *auth* are discriminatory for the SECURITY component. Term *site* is discriminatory for the component P2. Terms *framework* and *launcher* are frequent in bug reports belonging to the component FRAMEWORK in contrast to other components.

3.4 Component Reassignment Prediction

We perform experiments to examine the degree to which the derived statistical model (for automatic component categorization) can be used to make predictions for the task of predicting if an incoming bug report will be component reassigned or not. We apply the induced predictive model on a given bug report and compare the predictions with the initial component assigned by the bug reporter. If the best category (for computing Top 1 accuracy) does not match the initial component then we predict a component reassignment event. Similarly, for computing Top 2 accuracy we consider top two component predictions. We perform a series of experiments (for which the ground-truth is already available) and report the accuracy results of our predictor. The empirical results displayed in Table 10 indicate that Top 1 prediction accuracy (Eclipse project) of the proposed solution varies from 45.24% to 73.95%. Table 10 provide details

Table 9. Few examples of discriminatory features (terms) or important words discovered from the experimental dataset (Eclipse Project: JDT Product) to demonstrate correlation between terms and categories (components))

	TERM	APT	CORE	DEBUG	DOC	TEXT	UI
1	**file**	2.541689	1.219373	2.11113	1.580986	1	1.601105
2	**type**	25.42729	43.36233	13.08619	1	15.81292	18.3523
3	**junit**	25.42729	10.38253	4.005976	1	1.83097	14.06563
4	**compile**	10.66306	104.6105	1.869455	1	8.156138	8.171464
5	**swt**	1	44.14341	70.65426	1.219162	185.2767	96.19375
6	**workbench**	1	38.39945	40.04826	6.095812	108.1626	70.71629
7	**debug**	2.956659	7.652804	94.66307	1.201549	1	4.02394
8	**lib**	4.927765	5.765812	283.9892	6.007745	1	2.414364
9	**lang**	10.66306	25.38921	41.92921	1	14.1484	16.20897
10	**doc**	1	2.021024	6.837509	45.10901	1.420522	1.469854
11	**kit**	9.401176	1	21.42688	286.539	9.538987	1.535373
12	**jface**	1	37.33575	28.97801	1.219162	81.37564	59.28409
13	**text**	1	26.69879	12.69823	1.219162	70.41732	30.21366
14	**display**	1	22.01852	38.09469	3.657487	65.54696	54.05795
15	**action**	1	14.99812	14.6518	1.219162	46.47137	40.01268

results across two projects (Eclipse and Mozilla), 12 products and two classifiers (TF/IDF and Dynamic Language Model with NGRAM size = 6).

3.5 Component Reassignment Graphs

We conduct a series of experiments to understand component reassignment phenomenon from the perspective of network analysis. While the research goal of the work presented in Section 3.3 is to investigate the extent to which linguistic or textual features (sequence of characters, words, terms) present in bug reports can be used as discriminatory features for automatic component assignment, the goal of the work presented in this section is to view the component reassignment events as a graph or network and uncover patterns that can be used for the task of automatic component assignment. The proposal to construct component reassignment graph is based on our intuition and insight that there are some components which are more prone to reassignment and there is a correlation between certain components assigned by the bug reporter and the component to which the bug report is finally reassigned. We explain our idea using a concrete example (Figures 2) derived from Eclipse project experimental dataset. The bug report history contains information about initial component assignment (by the bug reporter) and subsequent component reassignments (if any) by the traiger or developers. We create a graph (called as the component reassignment graph) where nodes represent components and an arc from one node to another represents reassignment from one component to another. Thus the nodes represent components and arcs or links represent component reassignment relationship.

Table 10. [*Eclipse (ECL) and Mozilla (MZL)* Project] Accuracy results for the task of predicting if a component will be reassigned or not)

ECL PROD.	ML	TOP 1	TOP 2	MZL PROD.	ML	TOP 1	TOP 2
Core	TFIDF	52.19%	62.03%	Community	TFIDF	51.27%	67.28%
	DLM	50.52%	57.46%		DLM	45.24%	57.54%
Toolkit	TFIDF	54.76%	70.47%	Platform	TFIDF	52.55%	66.00%
	DLM	48.09%	60.00%		DLM	59.94%	72.72%
Firefox	TFIDF	59.09%	74.24%	BIRT	TFIDF	63.53%	73.32%
	DLM	60.35%	67.92%		DLM	67.75%	74.28%
Mozilla.org	TFIDF	59.34%	76.83%	Equinox	TFIDF	73.22%	89.00%
	DLM	63.62%	76.83%		DLM	72.34%	86.87%
Calendar	TFIDF	46.22%	62.22%	JDT	TFIDF	66.25%	75.00%
	DLM	44.88%	58.66%		DLM	68.12%	76.04%
addons.mozilla.org	TFIDF	65.56%	75.82%	PDE	TFIDF	68.54%	82.29%
	DLM	68.86%	77.28%		DLM	73.95%	83.12%

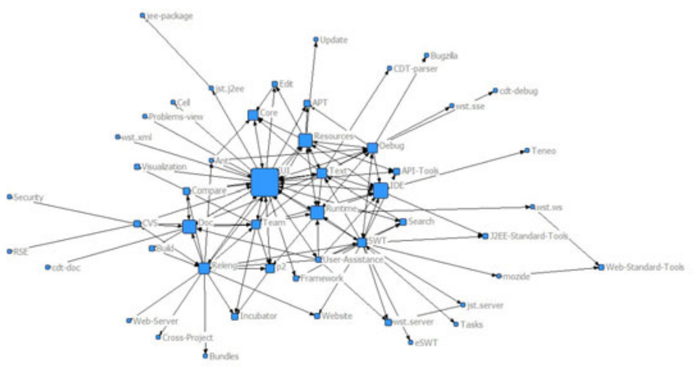

Fig. 2. A component reassignment graph (Eclipse project and Product Platform). The size of the node is proportional to the node's in-degree.

Figure 2 is a component reassignment graphs drawn using UCINET[4]) revealing the degree of reassignment relationship between components for a specific product (Eclipse Project and Product Platform).

Figures 2 is a directed graphs in which the out-degree (degree also referred to as valency) of a node (or a vertex) represents the number of times the component has been removed and the in-degree of a node represents the number of times the component (represented by the node) is added. In Figure 2, the size of the node is proportional to the node's in-degree. Figure 2 reveals that the in-degree (represented by the size of the node) for components like UI, IDE, Runtime, Doc and Resources is higher in contrast to other components. Table 11 shows that the reassignment relationship between pair of components varies and we believe

[4] `http://www.analytictech.com/ucinet/`

Table 11. Illustrative examples of number of times a component is removed (REM) and a different component is added or assigned (ADD) for Eclipse project and Platform product

	ADD	**REM**	**NUM**		**ADD**	**REM**	**NUM**
1	SWT	UI	59	7	Releng	P2	13
2	User Ast.	UI	41	8	SWT	IDE	11
3	UI	SWT	40	9	Team	UI	10
4	Text	UI	22	10	UI	Debug	10
5	SWT	Text	18	11	UI	Resources	10
6	Debug	UI	17	12	UI	Test	9

Table 12. Illustrative list of components (Eclipse, Platform) in increasing order of reassignment probability [P(RSN)]

Component	**Incorrect**	**Correct**	**Total**	**P(RSN)**
Compare	3	76	79	0.0379
Ant	2	38	40	0.05
Team	6	105	111	0.0540
Debug	19	168	187	0.1016
CVS	8	56	64	0.125
UI	208	967	1175	0.1770
Text	34	94	128	0.2656
Resources	24	60	84	0.2857
Runtime	24	23	47	0.5106
Website	12	2	14	0.8571

that this useful knowledge can be learned from historical data and can be used for making future predictions on automatic component reassignments. Table 12 is an Illustrative list of components (Eclipse Project, Platform Product) in increasing order of reassignment probability $P(RSN)$. Reassignment probability is the probability that the component will be reassigned or not and is computed as the ratio of two values: number of times the initially reported component is accurately classified (correct and no reassignment happens) to the number of the times the initially reported component is incorrectly classified (as a result of which a component reassignment event happens).

Figure 3 illustrates the architecture diagram for a system that combines a content-based (such as a TFIDF or a DLM based textual classifier) predictor with a link-based (component reassignment graph derived from historical data) predictor to make recommendation on the correct component or whether the assigned component will be reassigned or not. We perform experiments on Eclipse project data and observe that several times content-based classifier results in false positives (predicts a component to be reassigned when it should not be reassigned) outcomes. We believe that showing a score denoting the reassignment probability of component and the probability of a reassigning relationship with top k components can be useful to the developer.

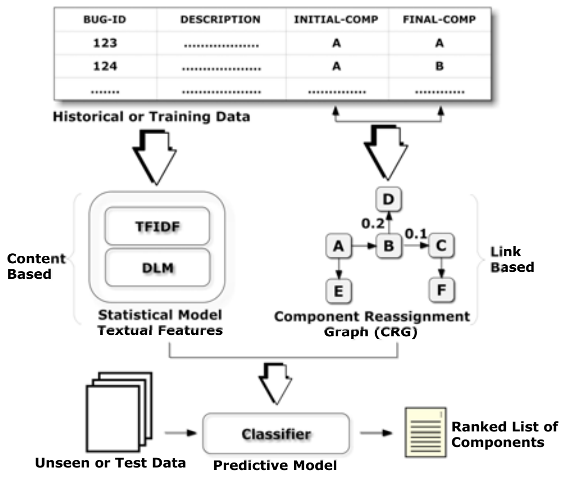

Fig. 3. Architecture diagram for the proposed automatic component reassignment system based on textual (terms present in bug reports) and link-based (derived from reassignment relationship) features

4 Summary

Empirical evidences demonstrate the presence of a phenomenon in which the initial component assigned by the bug reporter is later on reassigned during the life-cycle of a bug report. The study presented in this paper shows presence of correlation between terms in bug reports and components which can be exploited for the task of predicting the correct component of a bug report. We investigate the usefulness of machine learning based techniques for the task of predicting if a component reassignment event will occur or not given the initial component assigned by the bug reporter. Insights on component reassignment graphs and component reassignment probability are presented.

Acknowledgement. The work presented in this paper is partially supported by the Department of Science and Technology (DST, India) FAST grant awarded to the author. The author would like to acknowledge Sangeeta for her help in conducting experiments.

References

1. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: International Symposium on Foundations of software Engineering, SIGSOFT 2008/FSE-16, pp. 308–318. ACM, New York (2008)

2. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: Quality of bug reports in eclipse. In: OOPSLA Workshop on Eclipse Technology eXchange. ACM Press, New York (2007)
3. Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S.: Extracting structural information from bug reports. In: Working Conference on Mining Software Repositories, MSR 2008, pp. 27–30. ACM, New York (2008)
4. Bhattacharya, P., Neamtiu, I.: Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In: Inter. Conference on Software Maintenance, ICSM 2010, pp. 1–10. IEEE Computer Society, Washington, DC (2010)
5. Breu, S., Premraj, R., Sillito, J., Zimmermann, T.: Frequently asked questions in bug reports. Technical Report 2009-924-03, University of Calgary (March 2009)
6. Carpenter, B., Baldwin, B.: Natural Language Processing with LingPipe 4, draft edition. LingPipe Publishing, New York (2011)
7. Chen, L., Wang, X., Liu, C.: An approach to improving bug assignment with bug tossing graphs and bug similarities. JSW Journal of Software 6(3), 421–427 (2011)
8. Guo, P.J., Zimmermann, T., Nagappan, N., Murphy, B.: "not my bug!" and other reasons for software bug report reassignments. In: Computer Supported Cooperative Work, CSCW 2011, pp. 395–404. ACM, New York (2011)
9. Guo, P.J., Zimmermann, T., Nagappan, N., Murphy, B.: "Not My Bug!" and Other Reasons for Software Bug Report Reassignments. In: ACM Conference on Computer Supported Cooperative Work (2011)
10. Hooimeijer, P., Weimer, W.: Modeling bug report quality. In: IEEE/ACM International Conference on Automated Software Engineering, ASE 2007, pp. 34–43. ACM, New York (2007)
11. Jeong, G., Kim, S., Zimmermann, T.: Improving bug triage with bug tossing graphs. In: European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2009, pp. 111–120. ACM, New York (2009)
12. Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schröter, A., Weiss, C.: What makes a good bug report? IEEE Transactions on Software Engineering 36(5), 618–643 (2010)
13. Zimmermann, T., Premraj, R., Sillito, J., Breu, S.: Improving bug tracking systems. In: Companion to the 31th International Conference on Software Engineering (May 2009)