

Automated Bug Assignment in Software Maintenance Using Graph Databases

Satish C J

School of Computer Science and Engineering, VIT University, Vellore, 632014, India
E-mail: satish.cj@vit.ac.in

Anand Mahendran

School of Computer Science and Engineering, VIT University, Vellore, 632014, India
E-mail: manand@vit.ac.in

Received: 28 May 2017; Accepted: 21 July 2017; Published: 08 February 2018

Abstract—Processes involved in maintaining a system play a crucial role in enhancing customer satisfaction and longevity of the system. Maintenance engineers are the most critical resources in Software Maintenance. They play a significant role in fixing bugs and ensuring the normal functioning of systems. Software maintenance is a tedious task for novice engineers who are new to the system domain. The lack of up-to-date documentation makes system comprehension more challenging for inexperienced engineers. Assignment of high priority bugs to novice engineers may lead to inappropriate fixes and delay in the revival of an impacted system. Such issues may degrade customer satisfaction and also poor fixes can have a severe impact on the functioning of the system at a later stage. Our research is focussed on identification of engineers with the right level of experience to fix a given bug. We have used the concept of page ranking and graph databases to compute the importance of bugs and assignees in a graph. A newly reported bug will be scored and matched with bugs that have a similar score in the graph database. Assignees who have fixed a bug that closely maps the score of the reported bug will be assigned the task of fixing the bug. We have implemented this methodology using bug reports from QT framework on neo4j graph database. Our results are promising and will definitely pave way for a new bug assignment strategy in software maintenance.

Index Terms—Software maintenance, Software engineering, Bug Assignment, Graph Databases.

I. INTRODUCTION

Novice engineers who are assigned to maintenance tasks spend a lot of time and effort in understanding the existing system before making changes to the system [1]. As documents get outdated and obsolete during the maintenance phase, software change management and comprehension becomes cumbersome for novice engineers [2-3]. When such inexperienced engineers are assigned high priority bugs there is a greater chance of erroneous fixes or fixes getting delayed due to the meagre

domain knowledge.

High priority bugs should be handled with utmost care and any delays or errors associated with high priority bugs may have a severe impact on the business processes of an organization. Such issues can be avoided if we are able to identify or rank engineers based on their expertise with bugs on a given domain. The expertise of the engineers can be determined by associating them with the types of bugs they have fixed in the past. Our research is focused on identification of engineers with the right expertise to fix a bug. By right expertise we mean the experience gained by every engineer in fixing bugs of a specific type in a specific component of a project. When a new bug of a specific type arises from a specific component in a project then that bug is assigned to the next available engineer who has the maximum experience in fixing such type of bugs from that component.

Manual identification of an engineer to fix a bug is a time taking process [4]. For changing the status of a bug from Unconfirmed to New in Mozilla took 26.1 days on an average by managers. One of the main reasons for such processing is found to be the time spent on manual verification of bugs and identification of suitable engineers to fix the bugs [5]. Hence faster identification of suitable engineers for fixing bugs will be a great performance booster for software maintenance [6].

Research has focused on application of Machine learning techniques towards bug assignment strategies [7-12]. The issues with machine learning techniques are they are semi-automated techniques and they do not deal with inactive developers and work load balancing while assignment of bugs. Machine learning techniques make bug assignment more complicated as they need special tools and people with expertise for data pre-processing, application and result analysis. Moreover the results become inaccurate when the data gets older and there is a constant need for data sync between the transaction processing systems and the data warehouses.

There is a need for consistent update of data to the training set to maintain accuracy of the prediction. Moreover the number of classes is very large in the datasets, for instance there are 584 assignees to which we

can assign bugs for QT open source software and therefore we need separate machines, tools and resources to export data and run our algorithms on the data as the Y multiclass classification is a slow process. Identification of engineers who can work on severe tickets using machine learning is difficult as the number of severe tickets reported for a component is comparatively less when compared to normal tickets and the datasets normally suffer from the class imbalance problem. Khatun et al established that associating key words extracted from bug reports of bugs fixed in the past with engineers who fixed them can help determine the expertise of the engineers [13]. An algorithm has been proposed for extraction of keywords and identification of developers who fixed the bugs. This approach will require continuous refinement when new projects, components are implemented and the list of keywords are not static. Moreover the keywords of a bug cannot be alone used to determine the engineer as the bugs with the same keywords can have different priorities. A severe bug should be handled by an experienced engineer and a low priority bug should be assigned to a novice engineer. To overcome all these issues we propose graph databases as the solution towards maintenance of bug reports.

Our approach is unique and this is the first attempt towards using graph databases for scoring and assigning bugs to engineers. The method is preferable over the machine learning techniques as there is no process of extraction, parsing or syncing of data needed. The entire database for bug management can be on the graph database and our method can be effectively implemented on the bug management tool directly. The proposed method handles automated bug assignment along with workload balancing for engineers. Graph databases manage the data as graphs internally and hence make the relationships available as readymade graphs in the database. The identification of suitable maintenance engineers is possible with a few cypher queries without any specialized tools or extraction process. The method also offers a greater flexibility with respect to identification of engineers who map closely to the domain of a reported bug. This paper is organized as follows: in Section II, we have detailed our proposed methodology, in Section III we have discussed the implementation of the methodology and its results on an open source bug repository, in Section IV we have presented our conclusion and future work.

II. PROPOSED METHODOLOGY

The proposed methodology involves the following steps

A. Conversion of Projects, Components, Bugs and Assignees to Nodes in a graph using graph databases

The data on projects, components and their corresponding bugs are converted to nodes in a graph using a graph database. Separate nodes are created for projects, components, bugs and assignees in the database. Assignees are the engineers who are assigned bugs.

Project name, project id, project weight become properties of the project node. Component name, component id, component weight and project name become properties of the component node. Bug id, priority, assignee name, component name become the properties of the component node. Assignee id, assignee name become the properties of the assignee node. There can be addition of any other properties to the nodes but we recommend the addition of the properties mentioned above as a minimum requirement for the creation of the graph.

B. Assigning relationships between Projects, Components, Bugs and Assignees in the database

Relationships should be established between all the nodes created in the previous step. We have mapped all the components for a project using the project name in the project node and the project name in the component node. Bugs are mapped to components using the component name property in the bug node. Assignees are mapped to bugs using the assignee name property in the bug node.

C. Assignment of Initial Weights to Projects

Projects are created in every organization for implementation of new systems or for releasing a major revision to an existing system. Projects arise out of business demands and significance of a project can be determined by the importance of the business processes that are being automated. Every project should be assigned a weight based on the business processes supported. Such weights can be determined by business managers for an organization. We recommend the assignment of initial project weight (IP_{i^w} - Initial Project Weight for i^{th} project in the database) to every project in an organization

D. Assignment of Initial Weights to every Component in a Project

A Project is normally created for automating several business processes for an organization. Projects are made of components. Each component can in turn handle specific business functionality. Within a project each component can be assigned a weight based on the importance of the business functionality supported by the component. The business functionalities can be assigned weights after consulting the users of the functionality and business managers. Every component should be assigned initial component weights (IC_{i^w} - Initial Component Weight for i^{th} component in the database) within a project.

E. Assignment of Initial Weights to Bugs based on their Priority

Bugs can get their weight based on the priority they are assigned. Bugs are normally categorized as blockers, critical, important and low. Weights can be assigned to the bugs based on the priority. A blocker can get more weight than a critical bug. Every bug is assigned initial bug weights (IB_{i^w} - Initial Bug Weight for i^{th} bug in the database) based on the priority of the bug.

F. Computing the Final Weights of Components, Bugs and Assignees in the Graph

According to Page Rank algorithm, the page rank of a page depends on the page ranks of the pages pointing to it [14]. Likewise the importance of a component can be computed by using the importance of a project that contains the component. The computation of the final

component weight for component 1 (FC_{1w}) is the sum of the initial component weight assigned by business managers for component 1 and the initial project weight (IP_{1w}) for the project pointing to component 1 in the graph. The Final Component weights of two components are shown in Figure 1.

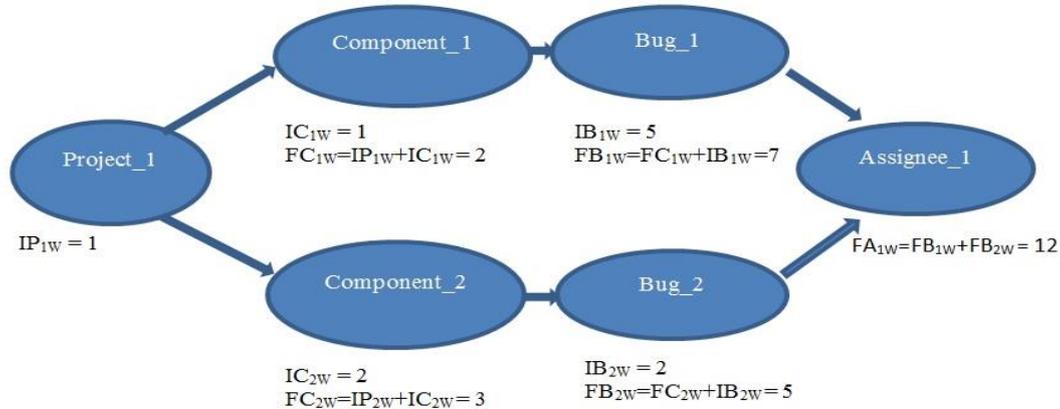


Fig.1. Computation of Final Weights in the Graph.

The computation of final bug weights (FB_{iw}) is done by summation of initial bug weight assigned using the priority of the bug with the final component weight computed for the component. The initial bug weights for bug1 and bug2 according to Figure 1 is 5 and 2. The final bug weight for bug 1 is the summation of Final component weight of component 1 and initial bug weight of bug 1. Likewise the process is repeated for computing the final bug weights for each bug in the database.

The weights are computed for assignees based on the computed final weight of the bugs linked to them in the graph. The weight for assignee 1 in Figure 1 is computed by summation of final bug weight for bug1 and bug2. This process is repeated for all assignee nodes in the graph. The components are graded using the projects they belong to along with their initial weights. The bugs are graded using the components that contain those bugs and the severity of the bugs. The assignees are graded using the type of bugs they have fixed. The list of assignees can be ordered by their weight to get a ranking of all assignees.

G. New Bug Assignment using Final Bug Weights and Final Assignee Weights in the graph

A new bug gets reported along with bug priority, the project name and component that contain the bug. Using the bug priority an initial bug weight (IB_{iw}) can be assigned to the bug. The project node and component nodes can be used for computing the final bug weight for the bug (FB_{iw}). Using the final bug score for the reported bug, the graph database should be scanned for bugs with similar FB_{iw} values belonging to the same component and project as that of the reported bug.

If the Reported bug's weight (FB_{iw}) = Database bug's weight (FB_{iw})

- Identify the assignee linked to the bug and assign the reported bug to that assignee.

If the Reported bug's weight (FB_{iw}) > All Database bugs weight (FB_{iw}) for that component in the project then

- Scan the bug database graph and find the bug with the highest level of FB_{iw} for the component
- Identify the assignee linked to that bug
- Assign the reported bug to that assignee

If the Reported bug weight (FB_{iw}) < All Database bugs weight (FB_{iw}) for that component then

- Scan the bug database graph and find the bug with the lowest level of FB_{iw} for that component in the project
- Identify the assignee linked to that bug
- Assign the reported bug to that assignee

If the Reported bug's (FB_{iw}) falls between the Fb_{iw} range of Database bugs [for instance if FB_{iw} of reported bug is 12 and we have bugs with FB_{iw} values ranging from 11 to 14 for that component in the database] then

- Scan the database graph to get the bug with an higher level of FB_{iw} than the reported bug for that component
- Identify the assignee for the bug in the graph
- Assign the reported bug to that assignee

H. Handling assignment overload for assignees

The maximum number of bugs an assignee should be assigned at any given time is denoted by Threshold Bug

Count (TBC). The TBC for all assignees will be determined by the team managers. The number of open bugs assigned to each assignee is denoted by the open bug count (OBC). Bugs should be only assigned to assignees whose OBC is less than TBC. If OBC is greater than or equal to TBC for a matching assignee then the next suitable bug in the list of database bugs that map to the reported bug's final bug weight can be determined. The assignee linked to that bug can be assigned to the reported bug after checking the TBC for that assignee. The process is repeated until an assignee with open bug count less than the threshold bug count is found. This strategy will prevent overloading assignees but it may lead to assignment of high priority bugs to inexperienced assignees when the experienced assignees are overloaded.

Consider the following graph database in figure 2 on bug reports. Node P represents Project; Node C1 represents a component of the project. Node B1 and B2 represent closed bugs on component C1. Node A1 and A2 represent assignees who have fixed the bugs B1 and B2. The following weights are assigned to nodes in the graph.

$$IP_{1w}(P) = 1 \quad (1)$$

$$IC_{1w}(C_1) = 2 \quad (2)$$

$$FC_{1w}(C_1) = IP_{1w}(P) + IC_{1w}(C_1) = 3 \quad (3)$$

$$IB_{1w}(B_1) = 3 \quad (4)$$

$$FB_{1w}(B_1) = FC_{1w}(C_1) + IB_{1w}(B_1) = 6 \quad (5)$$

$$IB_{2w}(B_2) = 2 \quad (6)$$

$$FB_{2w}(B_2) = FC_{1w}(C_1) + IB_{2w}(B_2) = 5 \quad (7)$$

$$FA_{1w} = FB_{1w}(B_1) = 6 \quad (8)$$

$$FA_{2w} = FB_{2w}(B_2) = 5 \quad (9)$$

$$TBC = 5 \quad (10)$$

$$OBC(A_1) = 2 \quad (11)$$

$$OBC(A_2) = 3 \quad (12)$$

Case 1:

If B_r is a newly reported bug with Final Bug weight (FB_{rw}) = 6 for project P and component C_1 . Then the newly reported bug's weight matches with weight of Bug B_2 in the database. assignee A_2 is linked to bug B_2 . The open bug count (OBC) of assignee A_2 is less than threshold bug count (TBC) and hence reported Bug B_r is assigned to assignee A_2 .

Case 2:

If the final bug weight (FB_{rw}) of $B_r = 8$ for project P and component C_1 then the bug with the highest FB_{iw} for that component in the database is found. The bug with the highest FB_{iw} in the graph for component C_1 is bug B_1 with a weight of 6. The assignee mapped to bug B_1 is assigned bug B_r .

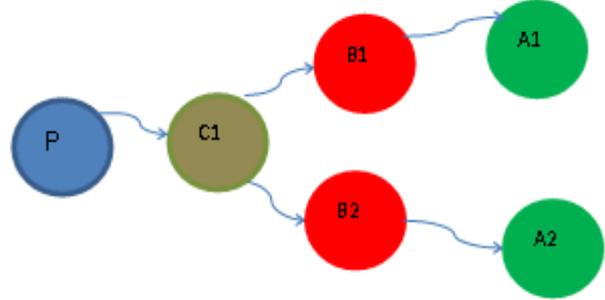


Fig.2. Sample Database Graph

If the OBC of assignee A_2 is greater than or equal to TBC then the bug with the next FB_{iw} is scanned for. Here bug B_1 is the next bug with FB_{iw} as 5. The reported Bug B_r is assigned to assignee of the bug B_1 if the OBC (A_1) is lesser than TBC. If all assignees are having OBC greater than or equal to TBC then the manager can take a decision on the assignment of the bug.

Case 3:

If FB_{rw} of $B_r = 4$ then the bug with the lowest FB_{iw} for that component in the graph is found. The bug with the lowest FB_{iw} in the graph for component C_1 is bug B_1 with a weight of 5. The assignee mapped to bug B_1 is assigned bug B_r . If OBC (A_1) greater than or equal to TBC then the bug with the next lowest FB_{iw} is scanned for. Here bug B_2 is the next bug for component C_1 with FB_{iw} as 6. The reported Bug B_r is assigned to assignee of the bug B_2 if OBC (A_2) is lesser than TBC. If all assignees are having OBC greater than or equal to TBC then the manager can take a decision on the assignment of the bug.

III. IMPLEMENTATION AND RESULTS

In this section we discuss about the implementation of our proposed methodology. The methodology was implemented using QT bug reports in neo4j graph database. QT is open source software. Bug management for QT is done using JIRA, a bug tracking tool developed by Australian Company Atlassian [15]. The implementation was carried out using the steps mentioned below.

A. Extraction of Bug Reports from JIRA

The bug reports were extracted from bug tracking tool JIRA as Excel Files. As JIRA was configured to allow only downloads of thousand bugs per report, there was a need to download many reports. Our approach was to download bug reports for every month, as the number of bugs reported for every month was lesser than thousand.

Reports for the last five years [2011-2016] were extracted as excel files from JIRA tool.

Using an excel macro to merge files; the individual reports were merged as one single excel file. The merged report had 43840 rows and 71 columns. Only bugs that had the status closed were considered in our implementation. There were a total of 28307 bugs with the closed status. The extraction process is shown in figure 3.

B. Loading Bug Report files in to Neo4j Graph database

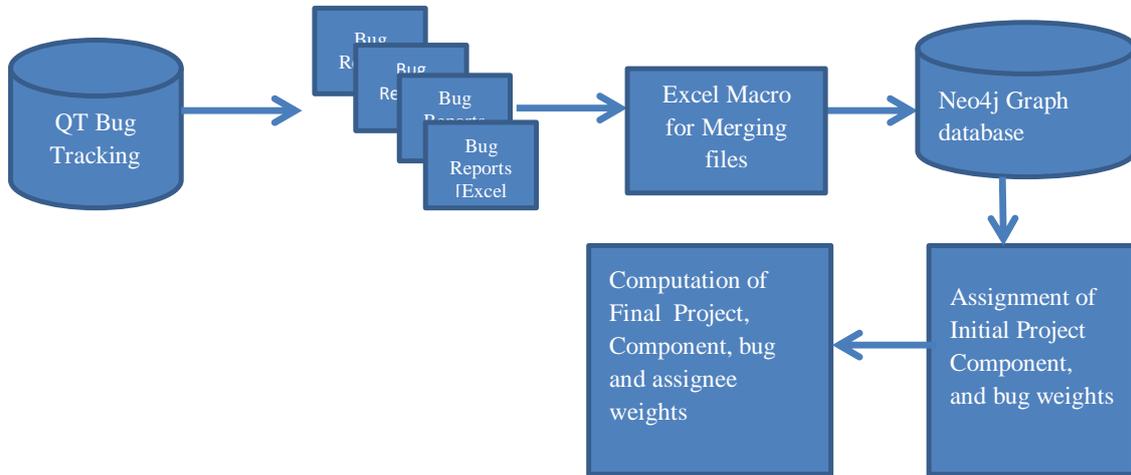


Fig.3. Extraction and Load Process

The following cypher query was used for extraction of project data and creation of project nodes in neo4j

- Load Csv With Headers From 'file:///project1.1.csv' AS Projects create(p:project{label: Projects. Project, id:Projects. Pid, pweight:Projects. Pweight})

The above query was modified and repeated for creation of components, bugs and assignees in neo4j

C. Creation of relationships between nodes

The relationships are created between projects and components using project name as a linking property in both project and component nodes. The relationship is named as contains. The cypher query for linking components with projects is given below

- match (p:project),(c:component) where p.label = c.project create (p)-[r:contains]->(c) return p,r,c;

Relationship between components and bugs is established using the component name property as a linking property for both the nodes. The relationship is named as sourceof. The cypher query for the creation of the relationship is given below

- match(c:component),(b:bug) where c.label = b.component create (c)-[r:sourceof]->(b) return b,r,c

The information on projects, components, bugs, assignees were extracted from the excel file and loaded in to neo4j graph database. We have assigned random initial weights to all projects in the database. There were 20 projects and each project was assigned a random weight between 1 and 10 as the initial project weight (IP_{iw}). The total number of components was 1875 for all projects. Random weights between 1 and 10 was assigned to all components as the initial component weight (IC_{iw}).

The relationship established between a sample project “Qt Creator” and all its components is shown in Figure 4(a). Relationship established between a component and all its bugs are shown in Figure 4(b) Relationship between bugs and assignees were created using the assignee name property as the linking property in both the bug and assignee nodes. The relationship is named as assignedto. The cypher query for establishing this relationship is given below

- match(b:bug),(a:assignee) where b.assignee = a.assignee create (b)-[r:assignedto]->(a) return b,r,a

Relationship established between bugs and assignees is shown in figure 4(c). A subsection of the final complex graph with all relationships is shown in Figure 4(d)

D. Computing Final Component Weights for Components in the graph

The final weight of a component is the summation of the initial component weight and the initial project component weight. The summation is achieved using the cypher query on the database.

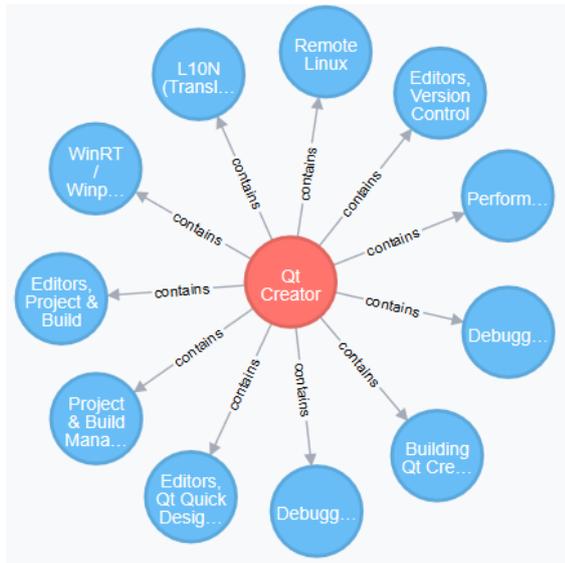
- match (p:project)-[r1:contains]->(c:component) set c.cweight = toint(p.pweight) + toint(c.cweight) return p,r1,c;

E. Computing Final Component Weights for all bugs in the graph

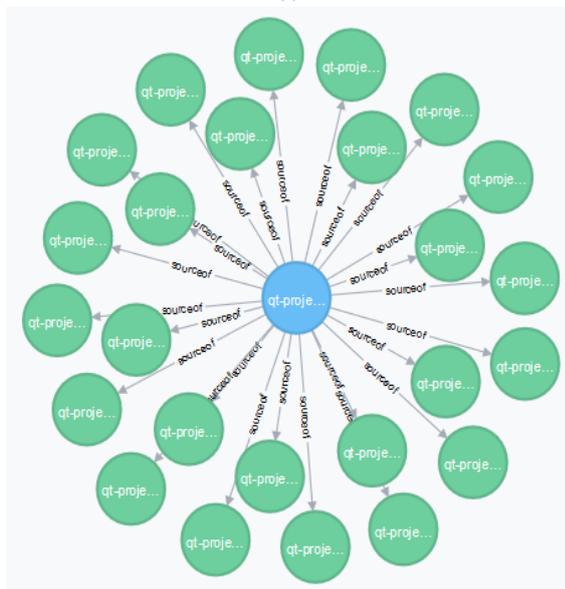
Initial bug weights (IB_{iw}) were assigned to bugs using the priority of bugs. Priority was maintained using the following priority levels for bugs reported for QT framework.

- P0: Blocker
- P1: Critical
- P2: Important
- P3: Somewhat important
- P4: Low
- P5: Not important

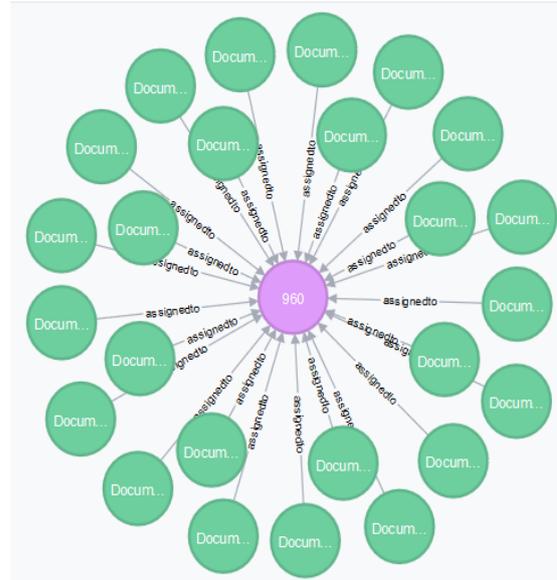
The initial bug weights assigned to bugs based on their priority is given in Table 1. We have assigned weights ranging from 0 to 6 in the descending order based on the priority. The initial bug weights can be determined by team leaders.



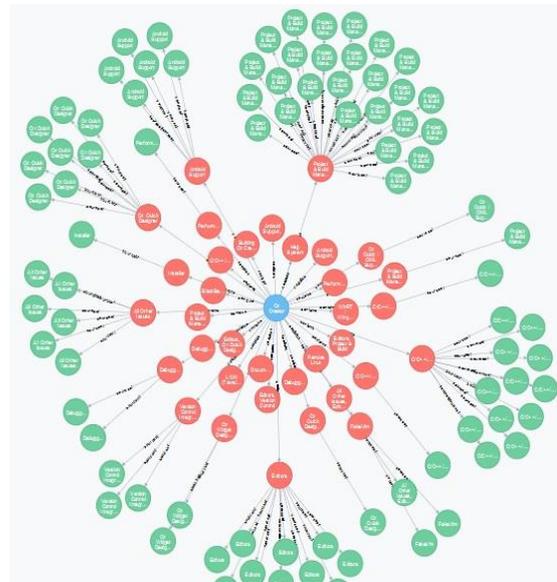
(a)



(b)



(c)



(d)

Fig.4. (a) - Relationship between Project and Component
 (b) - Relationship between Component and Bug
 (c) - Relationship between Bug and Assignee
 (d) - Subsection of Final Graph with all Relationships

Table 1. Initial Bug Weights

Bug Priority	Initial Bug Weights Assigned
P0: Blocker	6
P1: Critical	5
P2: Important	4
P3: Somewhat important	3
P4: Low	2
P5: Not important	1
Not Evaluated	0

The following Cypher queries were executed for assigning Initial Bug Weights

- `match (b:bug) where b.priority = 'Not Evaluated' set b.bweight = 0`
- `match (b:bug) where b.priority = 'P5: Not important' set b.bweight = 1`
- `match (b:bug) where b.priority = 'P4: Low' set b.bweight = 2`
- `match (b:bug) where b.priority = 'P3: Somewhat important' set b.bweight = 3`
- `match (b:bug) where b.priority = 'P2: Important' set b.bweight = 4`
- `match (b:bug) where b.priority = 'P1: Critical' set b.bweight = 5`
- `match (b:bug) where b.priority = 'P0: Blocker' set b.bweight = 6`

The final bug weight (FB_{iw}) is calculated by summing up the initial bug weight with the final component weight of the component which is the source of the bug. The following cypher query was executed on the database to compute final bug weights

- `match (c:component)-[r1:sourceof]->(b:bug) set b.bweight=toInt(c.cweight)+toInt(b.bweight) return c,r1,b`

F. Computing Final Assignee Weights for all Assignees in the Graph

The final weight for all assignees is computed as the sum of the final weights of all bugs linked to them in the graph. The following cypher query computes the weights of all assignees

- `match p=(b:bug)-[r:assignedto]->(a:assignee) set a.aweight=toint(b.bweight)+toint(a.aweight) return p`

The assignees can also be ordered based on the assignee weight using the cypher query below

- `match (a:assignee) return a.aweight,a.assignee order by a.aweight desc`

Table 2. Assignee Ranking

a.aweight	a.assignee
15877	Daniel Teske
11859	Tobias Hunger
10874	Friedemann Kleint
9221	Eskil Abrahamsen Blomfeldt
8779	Joerg Bornemann
8253	hjk
7893	Oswald Buddenhagen
6749	Thiago Macieira
6609	Eike Ziller
6021	J-P Nurmi

The ranking of the top 10 assignees that have the highest weights is given in table 2. This enables the tracking of experienced assignees in an organization

The approach offers a lot of flexibility. The experience gained by an assignee in a specific project or on a specific component or bug type can be computed. This flexibility allows identifying assignees with experiences from different perspectives within the database. If we wish to rank the assignees only based on the experience they have gained by working on high priority tickets [P0: blockers] in the database then we have to traverse through every bug with priority has P0: blockers that are linked to the assignee in the graph. The final assignee weight will be the sum of all final bug weights of bug nodes with priority as P0: blockers. We have executed the following cypher query on the database to find the assignees ordered by the experience they have gained by fixing only P0: blocker bugs.

The cypher query below identifies assignees having a relationship with blocker bugs in the graph and computes the final assignee weight by summing up the final bug weights of all blocker bugs linked to them. The final assignee weight gained for fixing blocker bugs is stored in the variable `prweight` for each assignee.

- `match (b:bug{priority:'P0: Blocker'})-[r3:assignedto]->(a:assignee) set a.prweight=toInteger(b.bweight)+ toInteger(a.prweight) return b,r3,a`

The assignees can then be ordered by the `prweights` using the cypher query below

- `match (a:assignee) return a.prweight,a.assignee order by a.prweight desc`

The result of the query is given in table 3. Likewise the ranking can be done for all assignees within a project, for a component, for a specific bug priority and so on. This method of ranking enables us to identify the candidates with varying levels of experience on various components, projects and types of bugs

Table 3. Assignee Ranking Based on Experience Gained by Fixing P0 Blocker Bugs

a.prweight	a.assignee
243	Iikka Eklund
216	Kai Khne
195	Oswald Buddenhagen
168	Heikki Halmet
162	Simon Hausmann
156	Daniel Teske
149	Eskil Abrahamsen Blomfeldt
145	Qt Release Team
144	Tor Arne Vesttb
130	Sean Harmer

G. Assignment of New Bugs to Assignees using the computed weights.

We have considered a specific project and component for bug assignment from the database. The name of the project is “Qt on Raspberry Pi (Obsolete)” and the name of the component mapped to that project is mt-cross-tools. The IP_{iw} assigned for the project is 3 and the IC_{iw} assigned for the component is 6 in the database. There are eight bugs mapped to the component as shown in figure 5. Assignee by name Rajiv M Ranganath has fixed and closed all the bugs for this component. The lowest FB_{iw} for the bugs is 11 and the highest FB_{iw} value is 15.

Case 1:

- If a new bug is reported for Component mt-cross-tools in project “Qt on Raspberry Pi (Obsolete)” with priority has P0: Blocker then the IB_{iw} value for the reported bug is 6 using table I values. The FB_{iw} for the bug is 15 [$FC_{iw} + IB_{iw}$].
- As the reported bug’s FB_{iw} value matches with the FB_{iw} bug in the database we retrieve the assignee [Rajiv M Ranganath] linked to the bug and assign the reported bug to Rajiv

The cypher query below retrieves the name of the assignee who has fixed bugs for component 'mt-cross-tools' that has a bug weight 15. The output of the query retrieves Rajiv and thus it is able to track the right assignee for the component.

- `match (b:bug{bweight:15,component:'mt-cross-tools'})-[:assignedto]->(a:assignee) return a.assignee,a.aweight`

Output of the query from neo4j is given in figure 6. We have not used OBC and TBC checks in our implementation as we have not loaded the bugs with the open status in to neo4j.

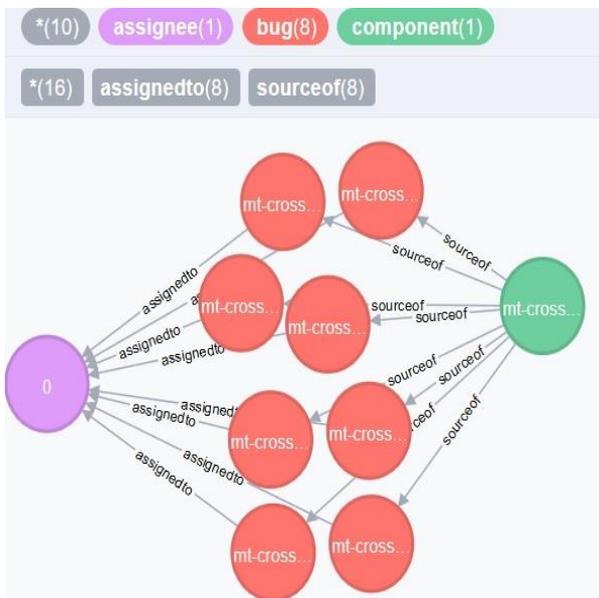


Fig.5. Bug database graph

```
$ match (b:bug{bweight:15,component:'mt-cross-tools');
```

Rows	Text
"a.assignee"	"a.aweight"
"Rajiv M Ranganath"	"92"

Fig.6. Assignee for Case 1 type Bug

Case 2:

If the reported bug has a bug weight greater than 15 then there will be no bugs matching the bug score of the reported bug. The assignee who fixed the bug with the highest final bug weight for the component will be assigned the bug. Let us consider a FB_{iw} of 18 for the newly reported bug. The match for a bug with FB_{iw} equal to 18 using the cypher query below will not return any nodes from the database

- `match (b:bug{bweight:18,component:'mt-cross-tools'})return b`

We will scan the graph for bugs with weights lesser or greater than the FB_{iw} of the reported bug using the cypher query given below. The output is given in figure 7.

- `match (b:bug{component:'mt-cross-tools'}) where b.bweight>18 or b.bweight<18 return b.id,b.bweight`

Since the reported bug has final bug weight greater than all closed bugs for the component, the new bug will be assigned to the assignee who fixed the bug with the highest FB_{iw} from the list. The assignee who fixed the bug with bug id 11415 will be assigned the newly reported bug. The cypher query below returns the assignee who has fixed the bug with the highest FB_{iw} in the list given in figure 7

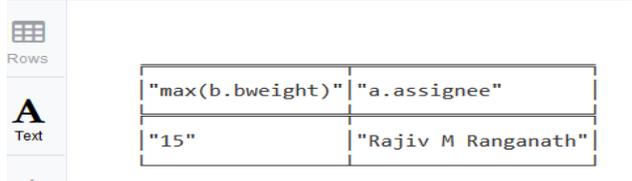
"b.id"	"b.bweight"
"2619"	"11"
"11415"	"15"
"11416"	"11"
"11417"	"11"
"15096"	"11"
"15097"	"11"
"21589"	"11"
"22279"	"11"

Fig.7. Bug weights <> 18

- `match (b:bug{component:'mt-cross-tools'})-[r1:assignedto]->(a:assignee) return max(b.bweight),a.assignee`

The assignee is assigned the newly reported bug and the output of the query is given in the figure 8 below

```
$ match (b:bug{component:'mt-cross-tools'})-[r1:as
```



"max(b.bweight)"	"a.assignee"
"15"	"Rajiv M Ranganath"

Fig.8. Assignee for Case 2 Scenario

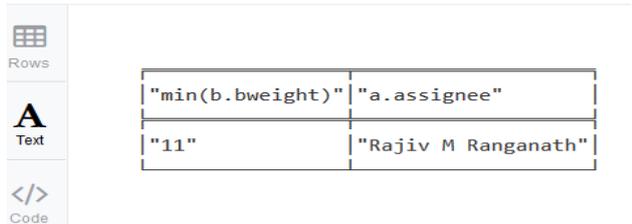
Case 3:

If the reported bug has a bug weight lesser than 11 then there will no bugs matching the bug score of the reported bug. The assignee who fixed the bug with the lowest final bug weight for the component in the graph will be assigned the bug. Let us consider an FB_{iw} of 10 for the newly reported bug. The bug will be assigned to the assignee who fixed the bug with the lowest FB_{iw} from the list given in figure 7. The cypher query is given below

- `match (b:bug{component:'mt-cross-tools'})-[r1:assignedto]->(a:assignee) return min(b.bweight),a.assignee`

The output of the query is given below in figure 9. Thus we were able to map the reported bugs to assignees with the right levels of experience using the final bug weights. The implementation of TBC and OBC checks can be achieved easily by importing bugs with open status in to the graph database and counting the number of bugs with the open status that are linked to an identified assignee. These checks can again be achieved by a few simple cypher queries.

```
$ match (b:bug{component:'mt-cross-tools'})-[r1:as
```



"min(b.bweight)"	"a.assignee"
"11"	"Rajiv M Ranganath"

Fig.9. Assignee for Case 3 Scenario

IV. CONCLUSION AND FUTURE WORK

Maintenance engineers are the life line to a software system. Reported bugs are assigned to engineers within a team without actually considering their levels of experience specific to the domain of the bug. This sort of assignment of bugs can be really disastrous when high

priority bugs get assigned to novice engineers who are new to the system. On the other hand low priority bugs should be assigned to novice engineers so that they get trained on the system without causing any major issues by faulty fixes. It is highly recommended that bugs are assigned to only engineers with the right levels of experience in the domain of the bug. The current assignment strategies do not consider the domain specific experience gathered by an engineer before assignment of the bug. Even an engineer who has gathered high levels of experience in fixing high priority bugs for one specific component in a project may not be suitable for fixing blocker bugs in another component for another project. Hence domain specific experience plays a very vital role in efficient bug fixes.

Our strategy of bug assignment is unique because it first assigns weights to projects, components and bugs and then assigns weights to engineers based on the experience they have gained by fixing bugs. We have converted the bug reports to graphs and identified the weights for every engineer using the weights of the bugs linked to them in the graph. This strategy not only scores engineers but also scores the bugs based on their sources of origin. Such a strategy effectively identifies engineers based on their bug fixes.

As we have utilized bug reports from open source software the identification of initial weights for projects and components was not done in consultation with business managers. We have assigned random weights for our projects and components and fixed weights for bugs in our experiment. The random weights may impact the correctness of the final results. Our approach also proves very effective when the number of projects, components, bugs and maintenance engineers are very high. Such a scenario exists for open source software. QT had 20 projects, 1875 components, 28307 bugs and 584 engineers. Such large data and relationships are very effectively handled in graph databases. The identification of domain specific expertise for such large number of engineers is also done effectively using the given strategy. Our future work will be on implementing a front end user interface for the bug assignment tool and extending our work to handle bug reassignment and reopening issues. We were able to demonstrate that bug assignment can be done using graph databases more effectively than the traditional database models by using actual bug reports. The results are highly promising and will definitely prove to be an area of interest for software maintenance engineers and researchers

REFERENCES

- [1] Khan, A. S., & Mattsson, M. K. "Management of documentation and maintainability in the context of software handover." In *Computing Technology and Information Management (ICCM), 2012 8th International Conference on IEEE*. Vol. 1, pp. 238-243, April 2012
- [2] Satish, C. J, and T. Raghuvveera. "Visualizing object oriented software using virtual worlds." In: *Proc. of the 4th WSEAS International Conf. on Software Engineering, Parallel & Distributed Systems*. World Scientific and Engineering Academy and Society (WSEAS), 2005.

- [3] Satish, C. J. and M. Anand. "Software Documentation Management Issues and Practices: A Survey." *Indian Journal of Science and Technology* Vol. 9, Issue 20, 2016.
- [4] Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K. and Cukic, B., "Automated triaging of very large bug repositories." *Information and Software Technology*, 2016
- [5] Jeong, G., Kim, S. and Zimmermann, T., "Improving bug triage with bug tossing graphs." *In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* pp. 111-120. ACM, 2009
- [6] Xia, Xin, et al. "Improving automated bug triaging with specialized topic model." *IEEE Transactions on Software Engineering* 43.3, pp 272-297, 2017.
- [7] Jonsson, L., Borg, M., Broman, D., Sandahl, K., Eldh, S. and Runeson, P., "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts." *Empirical Software Engineering*, 21(4), pp.1533-1578, 2016
- [8] Bhattacharya, P., Neamtiu, I. and Shelton, C.R., "Automated, highly-accurate, bug assignment using machine learning and tossing graphs." *Journal of Systems and Software*, 85(10), pp.2275-2292, 2012
- [9] Zhang, T. and Lee, B., March. "A hybrid bug triage algorithm for developer recommendation." *In Proceedings of the 28th annual ACM symposium on applied computing* pp. 1088-1094. ACM, 2013
- [10] Nagwani, N.K. and Verma, S., January. "Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes." *In ICT and Knowledge Engineering (ICT & Knowledge Engineering), 2011 9th International Conference on IEEE*, pp. 113-117, 2013
- [11] Tian, Y., Lo, D., Xia, X. and Sun, C. "Automated prediction of bug report priority using multi-factor analysis." *Empirical Software Engineering*, 20(5), pp.1354-1383, 2015
- [12] Zhang, W., Wang, S. and Wang, Q., "KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity." *Information and Software Technology*, 70, pp.68-84, 2016
- [13] Khatun, Afrina, and Kazi Sakib. "A bug assignment technique based on bug fixing expertise and source commit recency of developers." *Computer and Information Technology (ICCIT), 2016 19th International Conference on. IEEE*, 2016.
- [14] Page, L., Brin, S., Motwani, R. and Winograd, T., "The PageRank citation ranking: Bringing order to the web". *Stanford InfoLab*, 1999
- [15] QT issues download page <https://bugreports.qt.io/browse/QTWEBSITE-745?jql=>, Nov 2016

interests include Software Maintenance, Software Visualization and Software Documentation Management



Dr. Anand Mahendran received his Ph.D (Computer Science and Engineering) degree from VIT University, India in the year 2012, M.E (Computer Science and Engineering) degree from Government College of Engineering, Tirunelveli (Anna University), India in the year 2005 and B.E (Computer Science and Engineering) degree from VIT University, India in the year 2003. His

research interests include formal language theory and automata, bio-inspired computing models. He has published more than 35 papers in refereed international journals and refereed international conferences. He is currently working as an Associate Professor in School of Computer Science and Engineering, VIT University, Vellore, India.

How to cite this paper: Satish C J, Anand Mahendran, "Automated Bug Assignment in Software Maintenance Using Graph Databases", *International Journal of Intelligent Systems and Applications(IJISA)*, Vol.10, No.2, pp.27-36, 2018. DOI: 10.5815/ijisa.2018.02.03

Authors' Profiles



developing and maintaining software systems. His research

Satish C J is currently pursuing his Ph.D. degree with the School of Computer Science and Engineering, VIT University, Tamilnadu, India. He received his Master of Engineering degree from Anna University and Bachelor of Engineering degree from Madras University Tamilnadu, India. He was with Tata Consultancy Services for five years