BUG TRIAGING BASED ON TOSSING USING KNOWLEDGE GRAPH

Enroll. No. (s)- 19104004, 19104007, 19104010

Name of Student(s)- Amit G. Patil, Sanjoli Goyal, Muskan Jain

Name of Supervisor: Dr. Neetu Sardana

December- 2022

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

in

Information Technology

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGY JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

Table Of Contents

(II)

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and beliefs, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgment has been made in the text.

Place: Jaypee Institute of Information Technology, Sector 62, Noida Date: 10th December 2022	Signature: Name: Amit G. Patil Enrolment No: 19104004
Place: Jaypee Institute of Information Technology, Sector 62, Noida Date: 10th December 2022	Signature: Name: Sanjoli Goyal Enrolment No: 19104007
Place: Jaypee Institute of Information Technology, Sector 62, Noida Date: 10th December 2022	Signature: Name: Muskan Jain Enrolment No: 19104010

(III)

CERTIFICATE

This is to certify that the work titled “Bug Triaging Based on Tossing using KG” submitted by ―Amit G. Pati, Sanjoli Goyal and Muskan Jain in partial fulfillment for the degree of Bachelor of Technology of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of any other degree or diploma.

Signature of Supervisor:

Name of Supervisor: Dr. Neetu Sardana
Designation: Associate Professor
Date: 10th December 2022

(IV)

ACKNOWLEDGEMENT

We take this opportunity to express my heart-felt gratitude for our amazing supervisor Dr. Neetu Sardana, Associate Professor, Jaypee Institute of Information Technology, India for her consistent support, immense knowledge, scientific inputs and dedicated help. She has been exceptionally motivating, understanding and a guide in true sense.

Our special words of thanks should also go to Dr. Satish Chandra without the valuable guidance of whom this work would not have been possible. He has extended his valuable time, report insights and expert guidance during the evaluation process.

We would also like to thank Dr.Sharda Polwal for valuable feedback during the evaluation process.

We are also grateful to our university Jaypee Institute of Information Technology, Noidafor always providing timely technical support whenever needed and for providing excellent infrastructure and research environments.

Signature(s) of Students

Amit G. Patil (19104004)

Sanjoli Goyal(19104007)

Muskan Jain(19104010)

SUMMARY

For popular software systems, the number of daily submitted bug reports is high. Triaging these incoming reports is a time consuming task. Part of the bug triage is the assignment of a report to a dev with the appropriate expertise. In this project, we present an approach to automatically suggest devs who have the appropriate expertise for handling a bug report, based on the identified component obtained from the short description of the bug report. Our work is first to create the dev-label KG using text classification of issues and dev-reassignment data using tossing data. Using this data we create a heterogeneous graph consisting of labels and devs. We run Cypher Queries to return the ranked list of devs to recommend to a bug. We could say that our project recommends devs to a bug. A dev with who tosses less bugs is allotted higher rank in the list.

(VI)

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ACRONYMS

Abbreviation/ Symbol	Full Form	Description
KG	Knowledge Graph	It is a knowledge base that uses a graph-structured data model or topology to integrate data.
KNN	KNearest-Neighbor	K-Nearest Neighbour is one of the simplest ML algorithms based on Supervised Learning technique.
Dev	Developer	A developer is an individual that builds and creates software and applications.
ML	Machine Learning	Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior.
SNA	social network analysis	Social network analysis (SNA) is the process of investigating social structures through the use of networks and graph theory.

Chapter 1- Introduction

1.1 General Introduction

Bugzilla is a popular bug tracker used by many large projects, such as Mozilla, Eclipse, KDE, and Gnome. These applications receive hundreds of bug reports a day; ideally, each bug gets assigned to a dev who can fix it in the least amount of time. This process of assigning bugs, known as **bug assignment**, is complicated by several factors: if done manually, assignment is labor-intensive, time-consuming and fault-prone; moreover, for open source projects, it is difficult to keep track of active devs and their expertise. Identifying the right dev for fixing a new bug is further aggravated by growth, e.g., as projects add more components, modules, devs and testers, the number of bug reports submitted daily increases, and manually recommending devs based on their expertise becomes difficult.

A bug triage is basically finding the root cause of a bug. Where did it enter the code, how it entered and what can be done to prevent these kinds of bugs from being present in the future. A detailed analysis is done about the bug and necessary actions are taken to correct and prevent the bug.

For popular software systems, the number of daily submitted bug reports is high. Triaging these incoming reports is a time consuming task. Part of the bug triage is the assignment of a report to a dev with the appropriate expertise.

When a bug is assigned to a dev for the first time and if she is unable to fix it, the bug is assigned(tossed) to another dev. Thus a bug is **tossed** from one dev to another until a dev is eventually able to fix it. It can be inferred from the dataset of Eclipse that almost 90% of all "Fixed" bugs have been tossed at least once.

1.2 Problem Statement

Recently, bug fixing has become an important part of software maintenance. In large-scale projects, devs rely on bug reports to guide any bug-fixing activities. In our project, we are going to recommend a suitable dev who could resolve the bug in the most efficient way; such that the tossing length is minimized along with a knowledgeable dev who matches the requirements of solving a particular bug.

1.3 Significance of the problem

Due to a great number of bug reports submitted into the bug repository, the workload of the triagers who are responsible for arranging devs to fix the given bugs is very high. In order to reduce the triagers' workload, a number of approaches (e.g. ML algorithms and social network metrics) were proposed to study who should fix the bug report.

Surveys suggest that making the bug fixing process more efficient would reduce evolution effort and lower software production costs.

1.4 Empirical Study

Reports indicate that, on average, the Eclipse project takes about 40 days to assign a bug to the first dev, and then it takes an additional 100 days or more to reassign the bug to the second dev. Similarly, in the Mozilla project, on average, it takes 180 days for the first assignment and then an additional 250 days if the first assigned dev is unable to fix it. These numbers indicate that the lack of effective, automatic assignment and toss reduction techniques results in considerably high effort associated with bug resolution.

1.5 Brief Description of the Solution Approach

The basis of our work starts with using a graph based approach by creating a more visual notation of bugs and devs. We have three kinds of connections between bugs, skills and devs. The bugs and skills required to solve it, dev to dev showing who tossed the bug to whom, bugs initially assigned to which dev and the dev who solved it. We run Cypher Queries to return the ranked list of devs to recommend to a bug. A dev with who tosses less bugs is allotted higher rank in the list.

Chapter 2: Literature Survey

2.1 Summary of papers studied

Authors have observed that ML and information retrieval based bug assignment approaches are most popular in literature. A deeper investigation has shown that the trend of techniques is taking a shift from ML based approaches
towards information retrieval based approaches.[2]

In an earlier study on bug triage, D. Cubranic and G. Murphy proposed a Bayesian learning algorithm to find out who should fix a bug [10]. Their bug triage method can reduce the time required by manually analyzing bug reports. By executing an evaluated experiment, they found that the algorithm correctly predicted the most qualified dev to fix a new bug for approximately 30% of the bug reports. Anvik et al. continued the work of D. Cubranic and G. Murphy by determining the most appropriate dev for fixing a new bug [3]. When a new report comes, the proposed classifier produced by the machine-learning technique suggests a small number of devs suitable to fix the new bug. The experiment showed that the precision rate reached 57% for Eclipse projects and 64% for Firefox projects

Social network technique is a good way to capture additional features for implementing automatic bug triage. J. Xuan et al. modeled a dev prioritization in the bug reports of Eclipse and Mozilla bug repositories based on a social network technique [4]. By analyzing communication among devs in the comments, they ranked the devs based on their ability to accomplish three tasks, one of which being bug triage. W. Wu proposed an approach called DREX (Dev Recommendation with KNN Search and Experience Ranking) [5] which retrieves historical bug reports similar to the incoming bug and verifies the devs' experience according to the commenting activities in the social network. In DREX, they investigated various metrics including frequency and six social network metrics on the expertise ranking of DREX. They showed that DREX produced a better performance than the traditional text categorization method. T. Zhang et al. combined the dev’s experience and the fixing cost of historical bug reports [9]. As a result, they returned a ranked list of candidate devs. In our paper, we construct MDN which differs from the social network in the previous studies. The MDN can help to verify the devs' ability to fix the given bugs according to the number of received comments and sent commits. Compared with the social network, MDN does not only consider the comment activities, but also focuses on the commit messages for source code files change.

A question for us is how to prioritize the five devs for these two bug reports. This question is not easy to answer. Intuitively, both caniszczyk and cwindatt contribute to two bug reports while the other devs contribute to only one. On the other hand, ankur_sharma, cwindatt, and bcabe are very active since there are 3 links from each of them and 3 links to each of them (also a self-link for cwindatt). In this paper, we aim to model the dev prioritization. Furthermore, since dev factors are important for the metrics of software quality [10], another question is how to improve the software quality with the dev prioritization. In our work, we want to extract knowledge from the dev priorities to assist software development

We noticed that in OSS projects (see Section VI for related work), bug resolution is a collaborative work within a group of devs and social communities of devs with similar interest are arising from their frequent cooperation. Motivated by this observation, we introduce SNA here to discover those devs of great influence in devs' communities for bug resolution. Social network analysis [11] originated in sociology and is usually used to analyze the complex sets of relationships between members of social systems, with the goal of explaining social phenomena. The nodes in the social network being studied are a set of devs and the links of a social network are a set of relationships of these devs in contributing bug resolution. An essential function of SNA is to rank the importance of the devs according to their positions in the network [12]. To accomplish this goal, centrality indices are defined on the nodes of the graph, such as closeness centrality, graph centrality, betweenness centrality, etc. High centrality of a node means its relative high importance to other nodes. In this study, four metrics such as degree centrality, in-degree centrality and out-degree centrality, PageRank, betweenness centrality and closeness centrality are used to rank the devs' expertise. Degree centrality is defined as the number of links incident upon a node (i.e., the number of ties that a node has). Betweenness centrality is the number of shortest paths from all vertices to all others that pass through that node. Closeness centrality is defined as the mean geodesic distance (i.e., the shortest path) between a vertex v and all other vertices reachable from it. PageRank [13] is a variant of eigenvector centrality measure proposed by Sergey Brin and Larry Page, assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of measuring its relative importance within the set. It assigns relative scores to all nodes in the network based on the principle that connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes. Here, the description of the details of these social network metrics is out of the scope of this study. Readers who are interested in SNA can refer to [14] and [15].

2.2 Integrated summary of the literature studied

Authors have observed that ML and information retrieval based bug assignment approaches are most popular in literature.
By executing an evaluated experiment, they found that the algorithm correctly predicted the most qualified dev to fix a new bug for approximately 30% of the bug reports.
Murphy by determining the most appropriate dev for fixing a new bug. When a new report comes, the proposed classifier produced by the machine-learning technique suggests a small number of devs suitable to fix the new bug.
Social network technique is a good way to capture additional features for implementing automatic bug triage. modeled a dev prioritization in the bug reports of Eclipse and Mozilla bug repositories based on a social network technique.
By analyzing communication among devs in the comments, they ranked the devs based on their ability to accomplish three tasks, one of which being bug triage.
W. Wu proposed an approach called DREX (Dev Recommendation with KNN Search and Experience Ranking) which retrieves historical bug reports similar to the incoming bug and verifies the devs` experience according to the commenting activities in the social network.
In DREX, they investigated various metrics including frequency and six social network metrics on the expertise ranking of DREX. combined the dev`s experience and the fixing cost of historical bug reports.
In our paper, we construct MDN which differs from the social network in the previous studies.
The MDN can help to verify the devs' ability to fix the given bugs according to the number of received comments and sent commits. Compared with the social network, MDN does not only consider the comment activities, but also focuses on the commit messages for source code files change. A question for us is how to prioritize the five devs for these two bug reports.
Motivated by this observation, we introduce SNA here to discover those devs of great influence in devs` communities for bug resolution.
The nodes in the social network being studied are a set of devs and the links of a social network are a set of relationships of these devs in contributing bug resolution.
An essential function of SNA is to rank the importance of the devs according to their positions in the network.
In this study, four metrics such as degree centrality, in-degree centrality and out-degree centrality, PageRank, betweenness centrality and closeness centrality are used to rank the devs` expertise.
Here, the description of the details of these social network metrics is out of the scope of this study.

Chapter 3: Requirement Analysis and Solution Approach

3.1 Overall description of the project

3.1.1 Product Perspective

Recommending the suitable dev to resolve a particular bug is our main goal. Our dataset provides us with knowledge about bugs and all the devs who worked on that bug, along with the status of that bug. Usually a dev is assigned to a bug when his knowledge matches with the requirements of the bug. But in our project along with the skills we leverage the knowledge of previous tossing history of a dev; when assigning a dev to the bug. With this another variable into the recommendation algorithm we try to create a more credible bug assignment, minimizing the tossing length thus saving time and resources of our software company.

3.1.2 Product Function

To create a bug triager using SNA and KGs that helps in a more effective and efficient bug assignment process taking into account the knowledge as well as tossing behavior of the dev

3.1.3 Intended Audience and Reading Suggestions

Teachers, students and IT product based companies who desire to utilize this software for the betterment of their product whilst saving whatever resources it could when it comes to fixing bugs. Programmers and Professors who want to contribute to this software.

3.1.4 Product Scope

By observing the previous bug triaging data and using SNA, our triager makes more and more effective bug assignments taking into account skills and tossing behavior of the dev.

3.2 Requirement Analysis

3.2.1 Software Requirements

- Operating System: Windows
- VS Code or any other Python interpreting software
- Neo4j Graph Data Platform
- Supported Internet Browser: Chrome – Latest version

3.2.2 Hardware Requirements

- Processor: Intel i5 6 gen 64 bit
- RAM: 8 GB

- Hard disk Space: 20 GB
- An Internet Connection
- Keyboard and mouse or another pointing device

3.2.3 Libraries Used

- nltk
- json
- os
- Pandas
- Numpy

3.2.4 Language Used

- Python
- Cypher Query Language

3.3 Solution Approach

To find a suitable developer for a particular bug, our solution approach would involve the following steps:

1. Connect to the Neo4j database using the Python driver.
2. Define the nodes and relationships that you want to create in the graph using Cypher query language. This can include creating nodes for developers, bugs, priorities, components, and severities, as well as defining the relationships between these nodes.
Execute the Cypher queries using the Python driver to create the nodes and relationships in the graph.
3. Use additional Cypher queries, such as the one shown above, to query the graph and retrieve information about the relationships between the nodes. This can include finding the developers who have resolved bugs with the same priority, component, and severity as the given bug, as well as calculating the Adamic-Adar score for the relationship between the component node and the developer node.
4. Use the information retrieved from the graph to rank the recommended developers based on the number of bugs that they have resolved, the number of times that they have resolved bugs with the same severity as the given bug, and the number of times that they have tossed a bug to another developer. Developers who have resolved fewer bugs, have resolved more bugs with the same severity, and have tossed fewer bugs to other developers should be ranked higher.

5. Return the ranked list of recommended developers to the user.

By following these steps, we have a graph-based approach to recommend the most suitable developers for a particular bug, based on the component, priority, and severity of the bug, as well as the developers' past performance in resolving similar bugs. This approach can help to automate and improve the efficiency of the bug triage and assignment process.

Adamic Adar Score:

For Job Recommendation Adamic Adar metric is used. Adamic Adar is a measure used to compute the closeness of nodes based on their shared neighbors. This score is popularly being used in social networks for predicting future links. It is computed using the following formula:

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

where N(u) is the set of nodes adjacent to u.

A value of 0 indicates that two nodes are not close, while higher values indicate nodes are closer.

We have verified results using adamic adar closeness score which gives a non-zero score value while applying link prediction which basically means that the recommended job description is close to the resume.

Chapter 4: Modeling and Implementation Details

4.1 Design Diagrams

4.1.1 Use Case Diagram

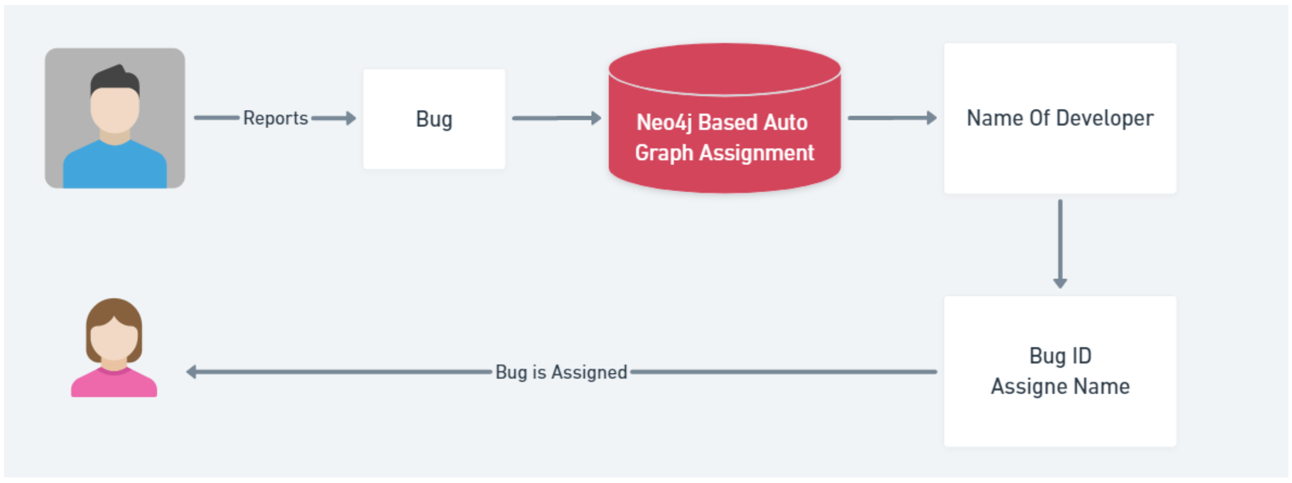


Fig 4.1

4.1.2 Control Flow Diagram

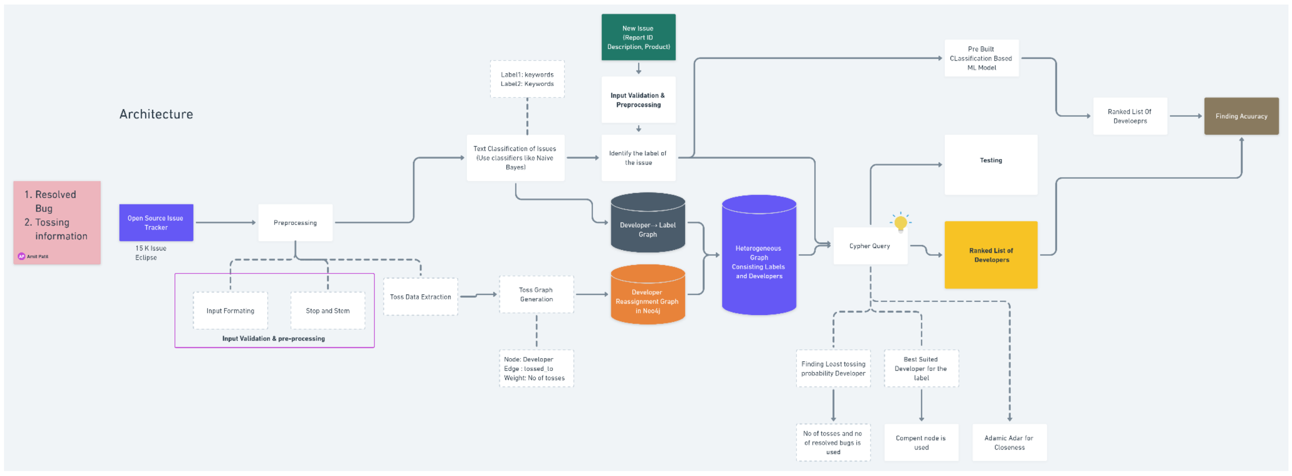

Architecture
1. Resolved Bug
2. Tossing information
Amit Patil
Open Source Issue Tracker
15 K Issue
Eclipse
Preprocessing
Input Formating
Stop and Stem
Input Validation & pre-processing
Toss Data Extraction
Toss Graph Generation
Node: Developer
Edge : tossed_to
Weight: No of tosses
Label1: keywords
Label2: Keywords
Text Classification of Issues (Use classifiers like Naive Bayes)
New Issue (Report ID Description, Product)
Input Validation & Preprocessing
Identify the label of the issue
Developer → Label Graph
Developer Reassignment Graph in Neo4j
Heterogeneous Graph Consisting Labels and Developers
Cypher Query
Pre Built CLassification Based ML Model
Ranked List Of Develoeprs
Finding Acuuracy
Testing
Ranked List of Developers
Finding Least tossing probability Developer
Best Suited Developer for the label
No of tosses and no of resolved bugs is used
Compent node is used
Adamic Adar for Closeness

Fig 4.2

4.2 Implementation Details and issues

4.2.1 Data Collection

Dataset is a collection of fixed bug reports gathered from an open source software bug tracker tool containing necessary information about the components, devs and re-assignments. This is categorized, classified, and semi-structured data. A bug report, generally a natural language text, submitted by the user is stored in the XML and JSON format by the bug tracker tool. Information contained in the dataset:

- severity: The severity denotes how soon the bug should be fixed.
- product: The particular software application the bug is related to.
- component: The relevant subsystem of the product for the reported bug.
- assigned to: The identifier of the dev to whom the bug was assigned to.
- short desc: Contains a natural language text embedded by the user.
- bug status: The status of the bug at every update. NEW, ASSIGNED, RESOLVED, VERIFIED, REOPENED.
- resolution: Tagging the bug report for maintenance. FIXED, REMIND, INVALID, WORKSFORME.

With this information the dependencies between the components, devs, and reassignment can be formed.

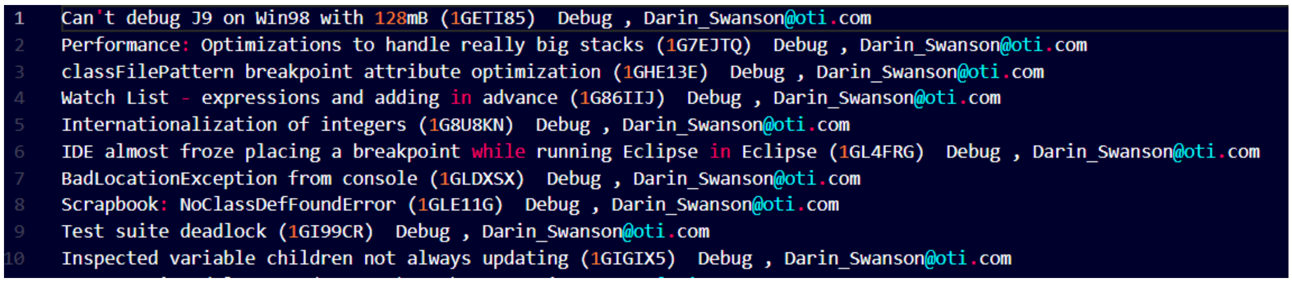
First the dataset in XML format is what we used but it has only around 10000 reports. To obtain higher efficiency, we used a dataset which is in JSON format. It has around 1,60,000 reports in a well structured manner.

4.2.3 Data Preprocessing

4.2.3.1 Formating short description of bug for further keyword extraction process

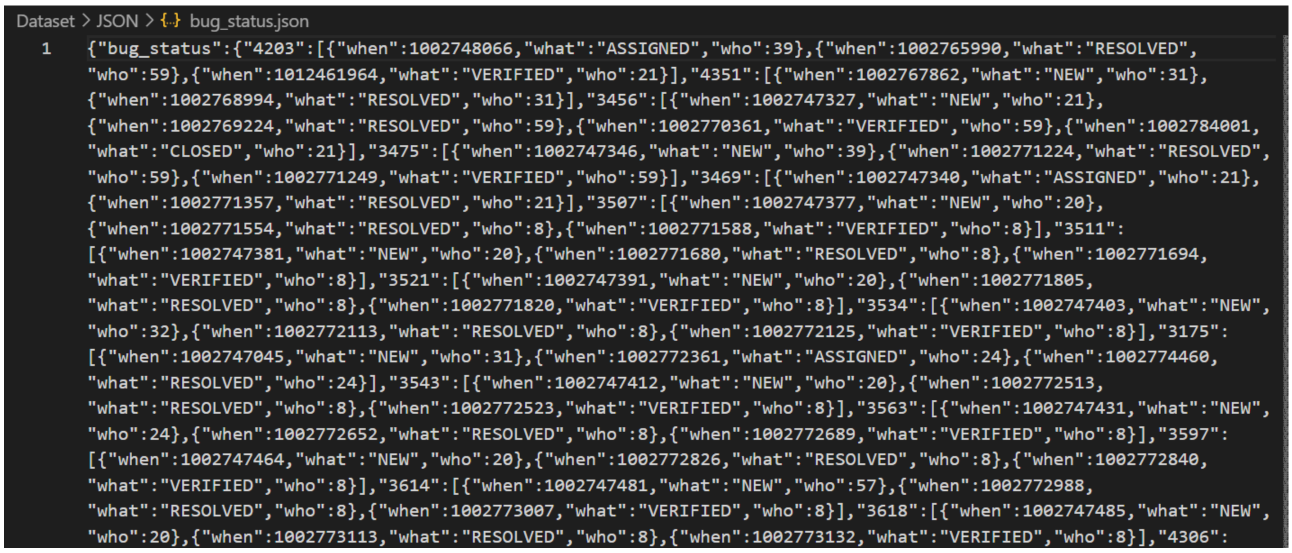
The "short_desc.json" file contains the timestamp, description and id of whom it was assigned to. "Component.json" file contains the component for the bug id and timestamp. The "assigned_to.json" file contains the details about the dev. We combined the information such that we have the short description, component and dev name mentioned in the .csv file format. We use this extracted information in the further steps.

Results:



```
Can't debug J9 on Win98 with 128mB (1GETI85)  Debug , Darin_Swanson@oti.com
Performance: Optimizations to handle really big stacks (1G7EJTQ)  Debug , Darin_Swanson@oti.com
classFilePattern breakpoint attribute optimization (1GHE13E)  Debug , Darin_Swanson@oti.com
Watch List - expressions and adding in advance (1G86IIJ)  Debug , Darin_Swanson@oti.com
Internationalization of integers (1G8U8KN)  Debug , Darin_Swanson@oti.com
IDE almost froze placing a breakpoint while running Eclipse in Eclipse (1GL4FRG)  Debug , Darin_Swanson@oti.com
BadLocationException from console (1GLDXSX)  Debug , Darin_Swanson@oti.com
Scrapbook: NoClassDefFoundError (1GLE11G)  Debug , Darin_Swanson@oti.com
Test suite deadlock (1GI99CR)  Debug , Darin_Swanson@oti.com
Inspected variable children not always updating (1GIGIX5)  Debug , Darin_Swanson@oti.com
```

Fig 4.3 Extraction Dataset



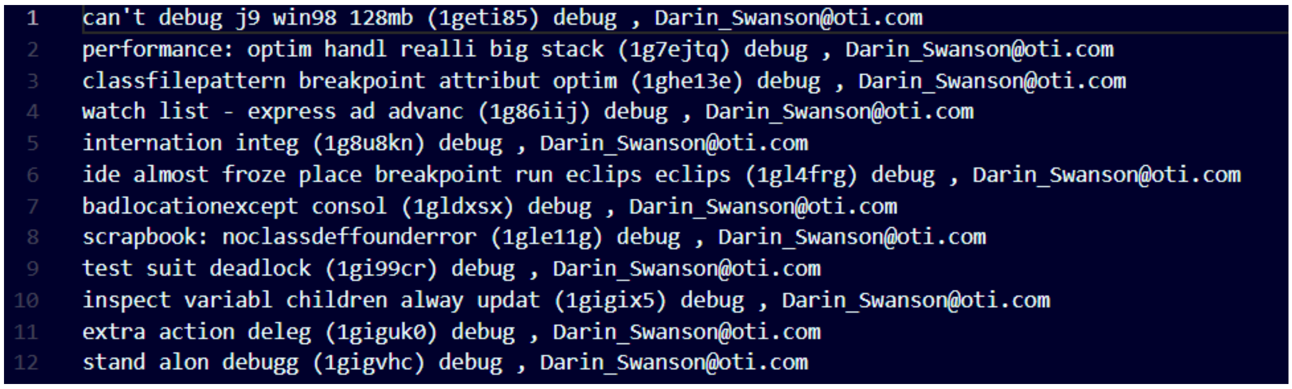
```
{"bug_status":{"4203":[{"when":1002748066,"what":"ASSIGNED","who":39},{"when":1002765990,"what":"RESOLVED",
"who":59},{"when":1012461964,"what":"VERIFIED","who":21}],"4351":[{"when":1002767862,"what":"NEW","who":31},
{"when":1002768994,"what":"RESOLVED","who":31}],"3456":[{"when":1002747327,"what":"NEW","who":21},
{"when":1002769224,"what":"RESOLVED","who":59},{"when":1002770361,"what":"VERIFIED","who":59},{"when":1002784001,
"what":"CLOSED","who":21}],"3475":[{"when":1002747346,"what":"NEW","who":39},{"when":1002771224,"what":"RESOLVED",
"who":59},{"when":1002771249,"what":"VERIFIED","who":59}],"3469":[{"when":1002747340,"what":"ASSIGNED","who":21},
{"when":1002771357,"what":"RESOLVED","who":21}],"3507":[{"when":1002747377,"what":"NEW","who":20},
{"when":1002771554,"what":"RESOLVED","who":8},{"when":1002771588,"what":"VERIFIED","who":8}],"3511":
[{"when":1002747381,"what":"NEW","who":20},{"when":1002771680,"what":"RESOLVED","who":8},{"when":1002771694,
"what":"VERIFIED","who":8}],"3521":[{"when":1002747391,"what":"NEW","who":20},{"when":1002771805,
"what":"RESOLVED","who":8},{"when":1002771820,"what":"VERIFIED","who":8}],"3534":[{"when":1002747403,"what":"NEW",
"who":32},{"when":1002772113,"what":"RESOLVED","who":8},{"when":1002772125,"what":"VERIFIED","who":8}],"3175":
[{"when":1002747045,"what":"NEW","who":31},{"when":1002772361,"what":"ASSIGNED","who":24},{"when":1002774460,
"what":"RESOLVED","who":24}],"3543":[{"when":1002747412,"what":"NEW","who":20},{"when":1002772513,
"what":"RESOLVED","who":8},{"when":1002772523,"what":"VERIFIED","who":8}],"3563":[{"when":1002747431,"what":"NEW",
"who":24},{"when":1002772652,"what":"RESOLVED","who":8},{"when":1002772689,"what":"VERIFIED","who":8}],"3597":
[{"when":1002747464,"what":"NEW","who":20},{"when":1002772826,"what":"RESOLVED","who":8},{"when":1002772840,
"what":"VERIFIED","who":8}],"3614":[{"when":1002747481,"what":"NEW","who":57},{"when":1002772988,
"what":"RESOLVED","who":8},{"when":1002773007,"what":"VERIFIED","who":8}],"3618":[{"when":1002747485,"what":"NEW",
"who":20},{"when":1002773113,"what":"RESOLVED","who":8},{"when":1002773132,"what":"VERIFIED","who":8}],"4306":
```

4.2.3.2 Stopword removal and Stemming Data

The short description of the bug has to be processed to extract the keywords. Stop-word and special Characters removal is first performed using the Natural Language ToolKit 8 9 in python. Then the same Toolkit is used to perform stemming on the text after the stop-words are removed.

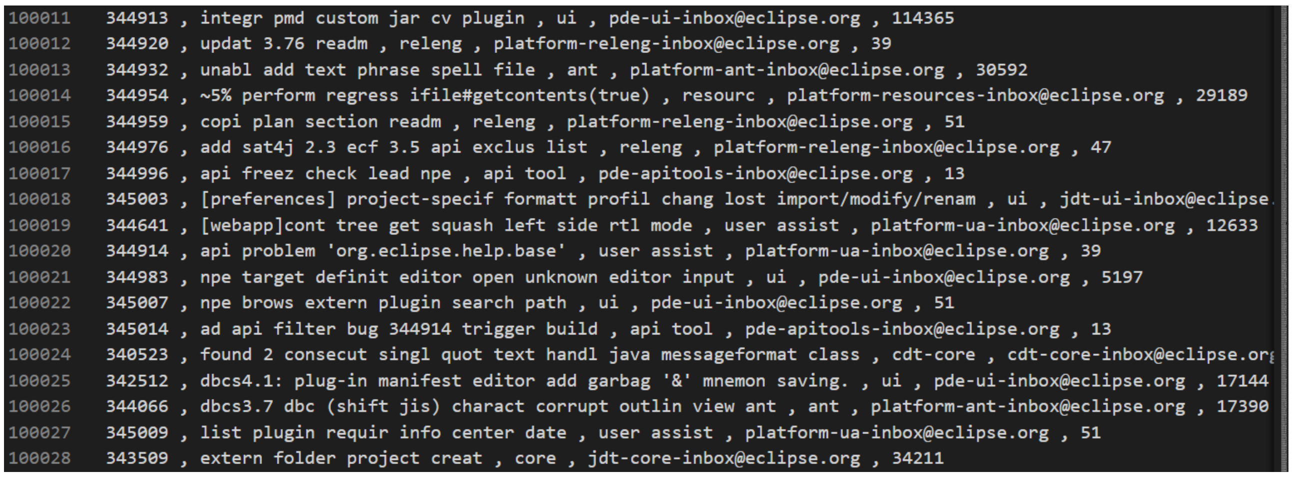
```python
def stem_and_stop(self, line):
        result = []
        for word in line.split():
            if word not in self.stop_words:
                result.append("%s " % self.stemmer.stem(word))
        return ''.join(result)
```

Results:



```
can't debug j9 win98 128mb (1geti85) debug , Darin_Swanson@oti.com
performance: optim handl realli big stack (1g7ejtq) debug , Darin_Swanson@oti.com
classfilepattern breakpoint attribut optim (1ghe13e) debug , Darin_Swanson@oti.com
watch list - express ad advanc (1g86iij) debug , Darin_Swanson@oti.com
internation integ (1g8u8kn) debug , Darin_Swanson@oti.com
ide almost froze place breakpoint run eclips eclips (1gl4frg) debug , Darin_Swanson@oti.com
badlocationexcept consol (1gldxsx) debug , Darin_Swanson@oti.com
scrapbook: noclassdeffounderror (1gle11g) debug , Darin_Swanson@oti.com
test suit deadlock (1gi99cr) debug , Darin_Swanson@oti.com
inspect variabl children alway updat (1gigix5) debug , Darin_Swanson@oti.com
extra action deleg (1giguk0) debug , Darin_Swanson@oti.com
stand alon debugg (1gigvhc) debug , Darin_Swanson@oti.com
```

Fig 4.4 Dataset after stemming and stop word removal



```
344913 , integr pmd custom jar cv plugin , ui , pde-ui-inbox@eclipse.org , 114365
344920 , updat 3.76 readm , releng , platform-releng-inbox@eclipse.org , 39
344932 , unabl add text phrase spell file , ant , platform-ant-inbox@eclipse.org , 30592
344954 , ~5% perform regress ifile#getcontents(true) , resourc , platform-resources-inbox@eclipse.org , 29189
344959 , copi plan section readm , releng , platform-releng-inbox@eclipse.org , 51
344976 , add sat4j 2.3 ecf 3.5 api exclus list , releng , platform-releng-inbox@eclipse.org , 47
344996 , api freez check lead npe , api tool , pde-apitools-inbox@eclipse.org , 13
345003 , [preferences] project-specif formatt profil chang lost import/modify/renam , ui , jdt-ui-inbox@eclipse
344641 , [webapp]cont tree get squash left side rtl mode , user assist , platform-ua-inbox@eclipse.org , 12633
344914 , api problem 'org.eclipse.help.base' , user assist , platform-ua-inbox@eclipse.org , 39
344983 , npe target definit editor open unknown editor input , ui , pde-ui-inbox@eclipse.org , 5197
345007 , npe brows extern plugin search path , ui , pde-ui-inbox@eclipse.org , 51
345014 , ad api filter bug 344914 trigger build , api tool , pde-apitools-inbox@eclipse.org , 13
340523 , found 2 consecut singl quot text handl java messageformat class , cdt-core , cdt-core-inbox@eclipse.or
342512 , dbcs4.1: plug-in manifest editor add garbag '&' mnemon saving. , ui , pde-ui-inbox@eclipse.org , 17144
344066 , dbcs3.7 dbc (shift jis) charact corrupt outlin view ant , ant , platform-ant-inbox@eclipse.org , 17390
345009 , list plugin requir info center date , user assist , platform-ua-inbox@eclipse.org , 51
343509 , extern folder project creat , core , jdt-core-inbox@eclipse.org , 34211
```

4.2.3.3 Extraction of Toss Data of Dev into .csv files

We used "assigned_to.json" and "component.json" files to accumulate Bug ID data along with the assigned dev name, tossed to the dev name till the name of the dev who resolved it.

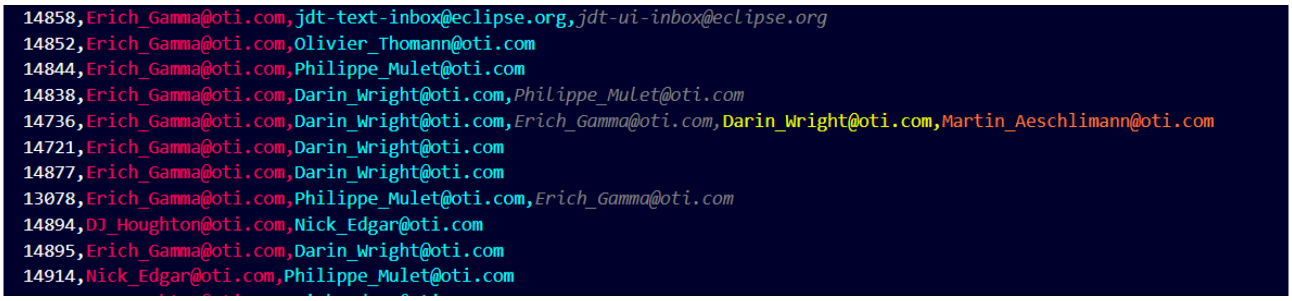
The component file contains all the instances of when the particular bug is assigned to which dev. While the assigned_to file contains information about dev id and dev name along with the timestamp of when the bug was assigned.

We compare both the time stamps and calculate the tossing data.

```
for bug_id, assignments in assigned_to.items():
        assignments_len = len(assignments)
        components_len = len(component[bug_id])

        line = bug_id + ","
        count = 0
        for i in range(0, assignments_len, 1):
            for j in range(0, components_len, 1):
                if assignments[i]["when"] == component[bug_id][j]["when"]
and assignments[i]["what"] is not None and \
                        component[bug_id][j]["what"] in (
                    "UI", "Core", "Text", "Debug", "APT", "Doc"):

                    if count != 0:
                        line += ","
                    line += (assignments[i]["what"])
                    count += 1
        if count >= 2:
            line += "\n"
            output_file.write(line.encode('utf-8'))
```

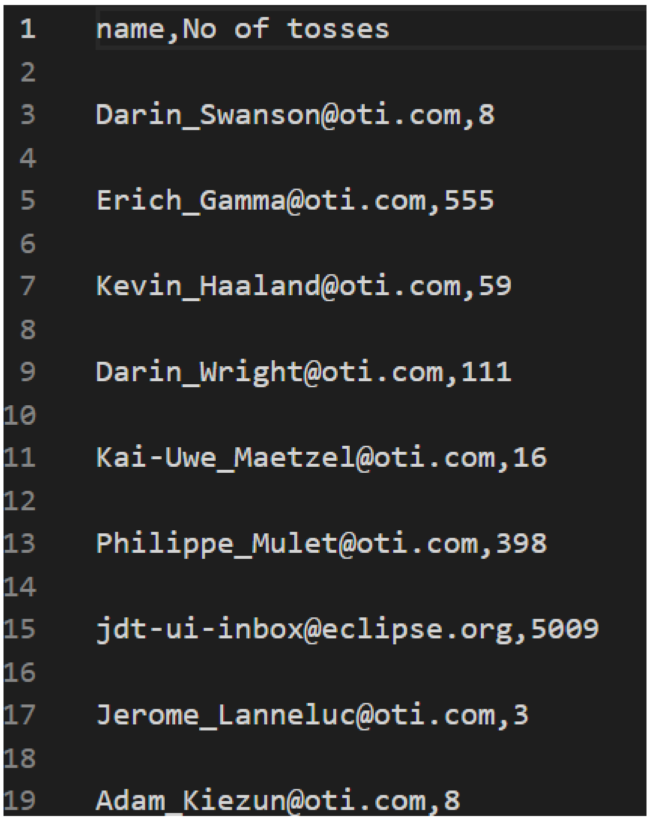
All the data is appended into a single string which is then transferred to the .csv file.

Results:



```
14858,Erich_Gamma@oti.com,jdt-text-inbox@eclipse.org,jdt-ui-inbox@eclipse.org
14852,Erich_Gamma@oti.com,Olivier_Thomann@oti.com
14844,Erich_Gamma@oti.com,Philippe_Mulet@oti.com
14838,Erich_Gamma@oti.com,Darin_Wright@oti.com,Philippe_Mulet@oti.com
14736,Erich_Gamma@oti.com,Darin_Wright@oti.com,Erich_Gamma@oti.com,Darin_Wright@oti.com,Martin_Aeschlimann@oti.com
14721,Erich_Gamma@oti.com,Darin_Wright@oti.com
14877,Erich_Gamma@oti.com,Darin_Wright@oti.com
13078,Erich_Gamma@oti.com,Philippe_Mulet@oti.com,Erich_Gamma@oti.com
14894,DJ_Houghton@oti.com,Nick_Edgar@oti.com
14895,Erich_Gamma@oti.com,Darin_Wright@oti.com
14914,Nick_Edgar@oti.com,Philippe_Mulet@oti.com
```

Fig 4.5 Toss data csv file dataset



```
name,No of tosses

Darin_Swanson@oti.com,8

Erich_Gamma@oti.com,555

Kevin_Haaland@oti.com,59

Darin_Wright@oti.com,111

Kai-Uwe_Maetzel@oti.com,16

Philippe_Mulet@oti.com,398

jdt-ui-inbox@eclipse.org,5009

Jerome_Lanneluc@oti.com,3

Adam_Kiezun@oti.com,8
```

4.2.3.4 Extraction of Skill set of individual Developers into .csv files

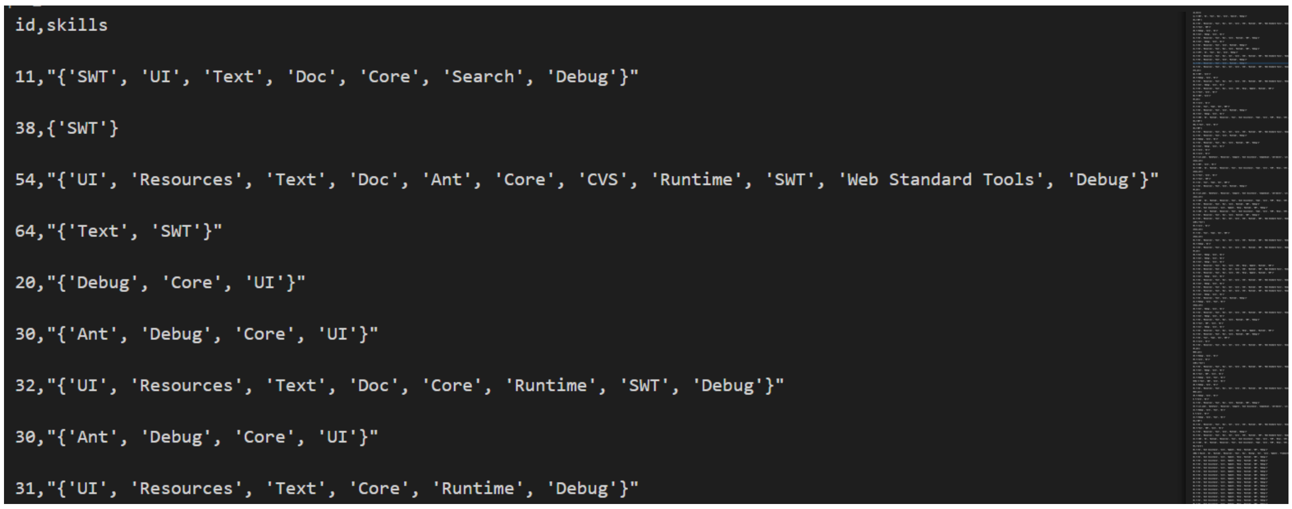
In this process, we considered that if a dev is able to resolve the bug then whatever the component of that bug is, it becomes the skill of the dev.

We start by using data from the previous file we created. We took the names of the last dev who had a particular bug. If that particular bug is resolved which we check from the status of the bug in the "resolution.json" file then we proceed to the next step. We use the "component.json" file to extract the data of the bug. This data becomes the skills the dev has who was able to resolve the bug.

```
def knowledge():
    dev_id= None
    skill_set=set()
    dictionary_dev = []
    dev_knowledge= defaultdict(set)
    for each in toss_data:
        dev_id= None
        data = each.split(',')
        dev = {}
        # check for bugid- data[0] in resolution
        for id in resolution:
            if(id==data[0] and resolution[id][-1]["what"] != ""):
                dev_id= resolution[id][-1]["who"]
                dev["id"] = dev_id

        # check for components of the resolved id
        if(dev_id!= None):
            for every in component[data[0]]:
                if(every["who"]== dev_id ):
                    dev_knowledge[dev_id].add(every["what"])
                    skill_set.add(every["what"])
            dev["skills"]= dev_knowledge[dev_id]
        dictionary_dev.append(dev)
```

We then transferred the data to .csv file in the form of dev id, dev skill.

Results:



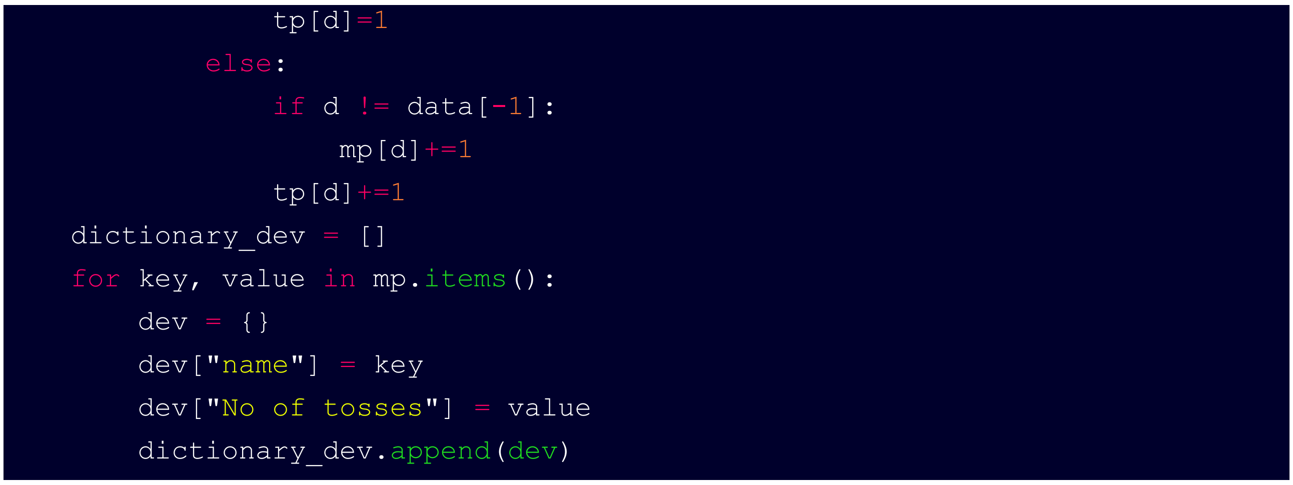
```
id,skills

11,"{'SWT', 'UI', 'Text', 'Doc', 'Core', 'Search', 'Debug'}"

38,{'SWT'}

54,"{'UI', 'Resources', 'Text', 'Doc', 'Ant', 'Core', 'CVS', 'Runtime', 'SWT', 'Web Standard Tools', 'Debug'}"

64,"{'Text', 'SWT'}"

20,"{'Debug', 'Core', 'UI'}"

30,"{'Ant', 'Debug', 'Core', 'UI'}"

32,"{'UI', 'Resources', 'Text', 'Doc', 'Core', 'Runtime', 'SWT', 'Debug'}"

30,"{'Ant', 'Debug', 'Core', 'UI'}"

31,"{'UI', 'Resources', 'Text', 'Core', 'Runtime', 'Debug'}"
```

Fig 4.6 Dev's skill csv file

4.2.3.5 Total tosses of a Dev into .csv file

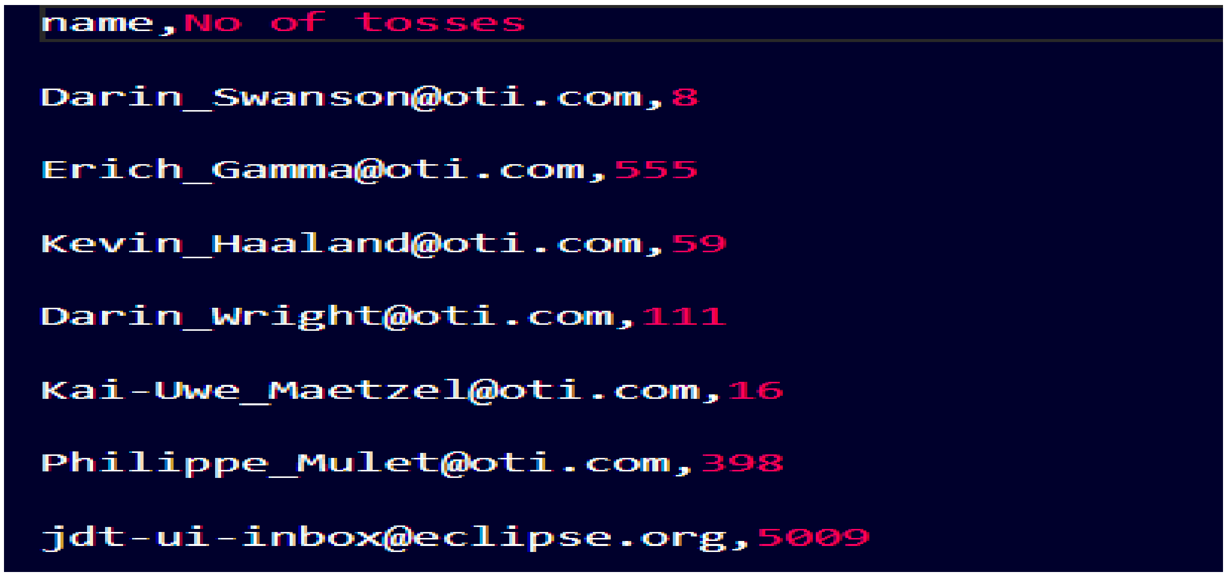
We calculated how many times a dev has tossed bugs. Then transferred our learning to the .csv file.

```
def tosses_more():
    l = []
    mp = dict()
    tp=dict()
    for each in toss_data:
        data = each.split()
        for d in data:
            if d not in l:
                l.append(d)
                mp[d] = 1
```



```
                tp[d]=1
            else:
                if d != data[-1]:
                    mp[d]+=1
                tp[d]+=1
    dictionary_dev = []
    for key, value in mp.items():
        dev = {}
        dev["name"] = key
        dev["No of tosses"] = value
        dictionary_dev.append(dev)
```

Results:

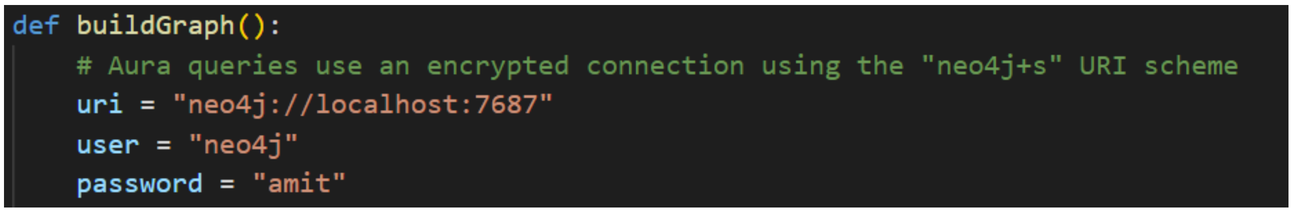


```
name,No of tosses

Darin_Swanson@oti.com,8

Erich_Gamma@oti.com,555

Kevin_Haaland@oti.com,59

Darin_Wright@oti.com,111

Kai-Uwe_Maetzel@oti.com,16

Philippe_Mulet@oti.com,398

jdt-ui-inbox@eclipse.org,5009
```

Fig 4.7 Dev's Tosses count

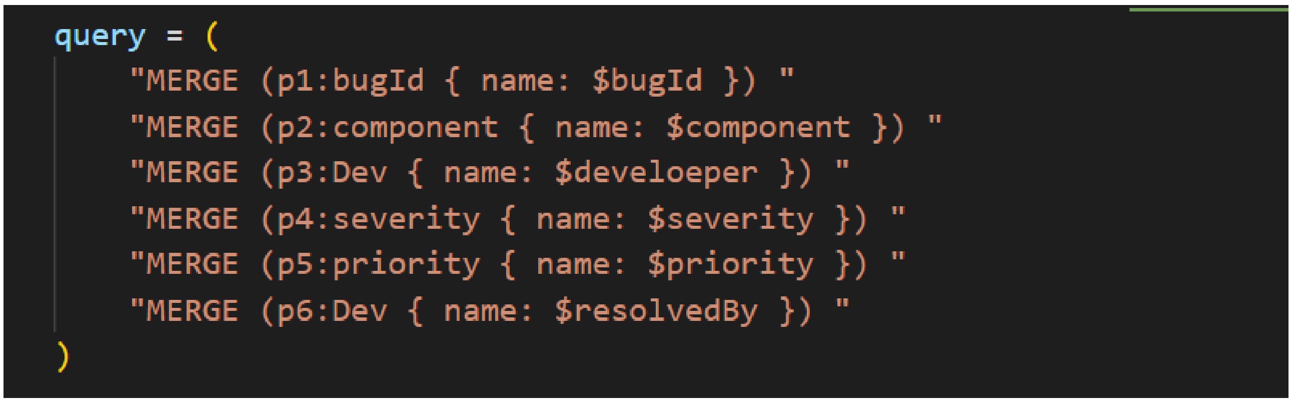
4.2.4 Building the KG

Connect to the Neo4j database using the Python driver.



```
def buildGraph():
    # Aura queries use an encrypted connection using the "neo4j+s" URI scheme
    uri = "neo4j://localhost:7687"
    user = "neo4j"
    password = "amit"
```

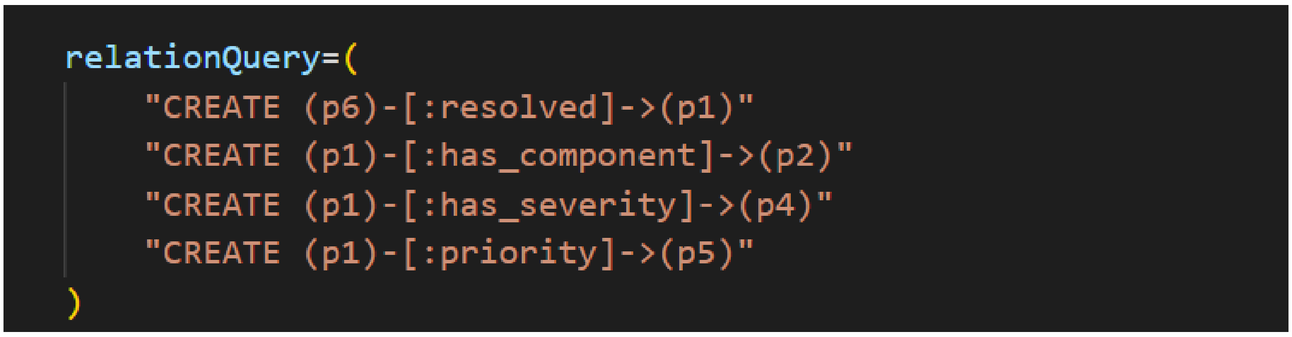
Defining the nodes and relationships to create in the graph using Cypher query language. This can include creating nodes for developers, bugs, priorities, components, and severities, as well as defining the relationships between these nodes.



```
query = (
    "MERGE (p1:bugId { name: $bugId }) "
    "MERGE (p2:component { name: $component }) "
    "MERGE (p3:Dev { name: $develoeper }) "
    "MERGE (p4:severity { name: $severity }) "
    "MERGE (p5:priority { name: $priority }) "
    "MERGE (p6:Dev { name: $resolvedBy }) "
)
```

Execute the Cypher queries using the Python driver to create the nodes and relationships in the graph.

Use additional Cypher queries to query the graph and retrieve information about the relationships between the nodes.

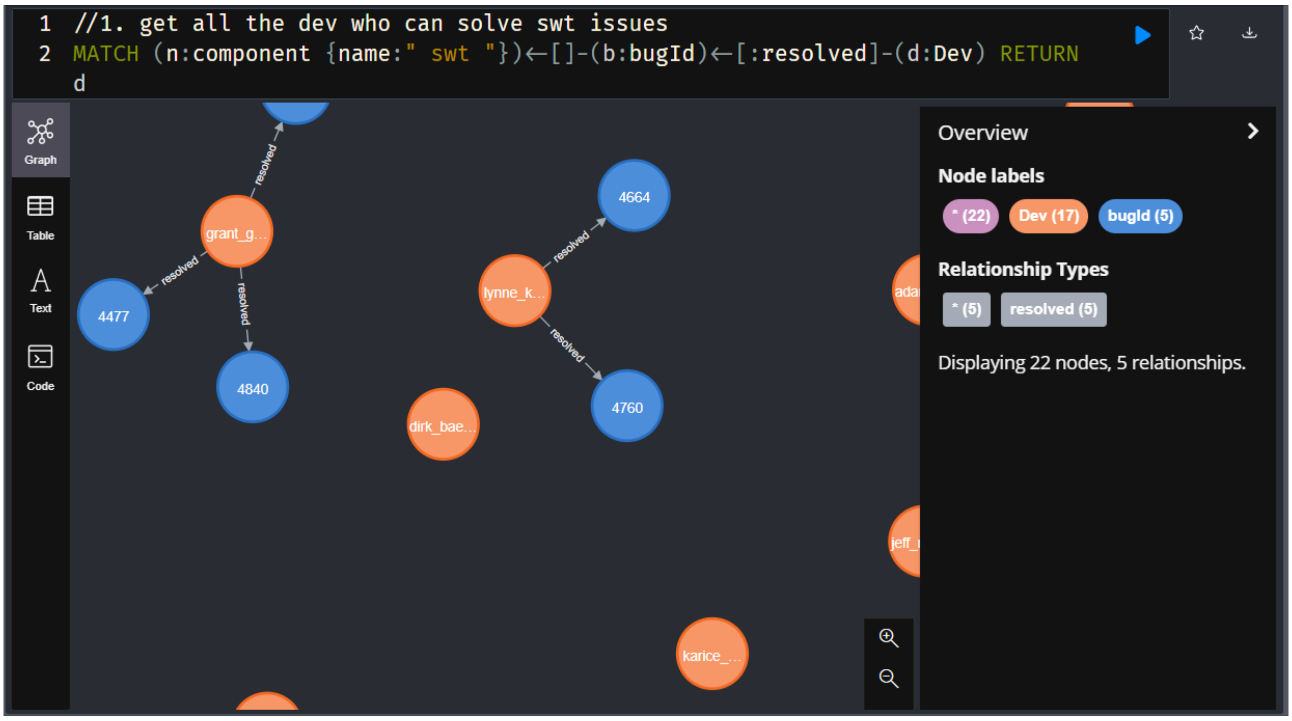


```
relationQuery=(
    "CREATE (p6)-[:resolved]->(p1)"
    "CREATE (p1)-[:has_component]->(p2)"
    "CREATE (p1)-[:has_severity]->(p4)"
    "CREATE (p1)-[:priority]->(p5)"
)
```

4.2.5 Cypher Queries Answered

4.2.5.1. get all the dev who can solve swt issues

This Cypher query is used to find all developers who have resolved bugs related to the "swt" component. The query first matches the component node, bug node, and developer node that are relevant to the "swt" component. It then returns the developer node, which represents the developers who have resolved bugs related to the "swt" component. This query can be used to identify the developers who have expertise in fixing bugs related to the "swt" component, which can be useful for assigning new bugs to the most suitable developers.



```
//1. get all the dev who can solve swt issues
MATCH (n:component {name:" swt "})←[]-(b:bugId)←[:resolved]-(d:Dev) RETURN
d
```

Displaying 22 nodes, 5 relationships.

Fig 4.8

4.2.5 .2. get all the dev who can solve "swt" issues considering priority and severity

This Cypher query is similar to the previous one, but it adds additional criteria to rank the recommended developers. The query first matches the component node, bug node, and developer node that are relevant to the "swt" component. It then matches the priority and severity nodes for each of these bugs, and uses these nodes to rank the recommended developers.

The developers are ranked based on the priority and severity of the bugs that they have resolved, with developers who have resolved higher priority and higher severity bugs being ranked higher. The results are then returned in descending order based on this ranking, so that the developers who have resolved the most critical bugs are returned first. This query can be used to identify the most experienced and skilled developers for fixing bugs related to the "swt" component, which can be useful for assigning new bugs to the most suitable developers.

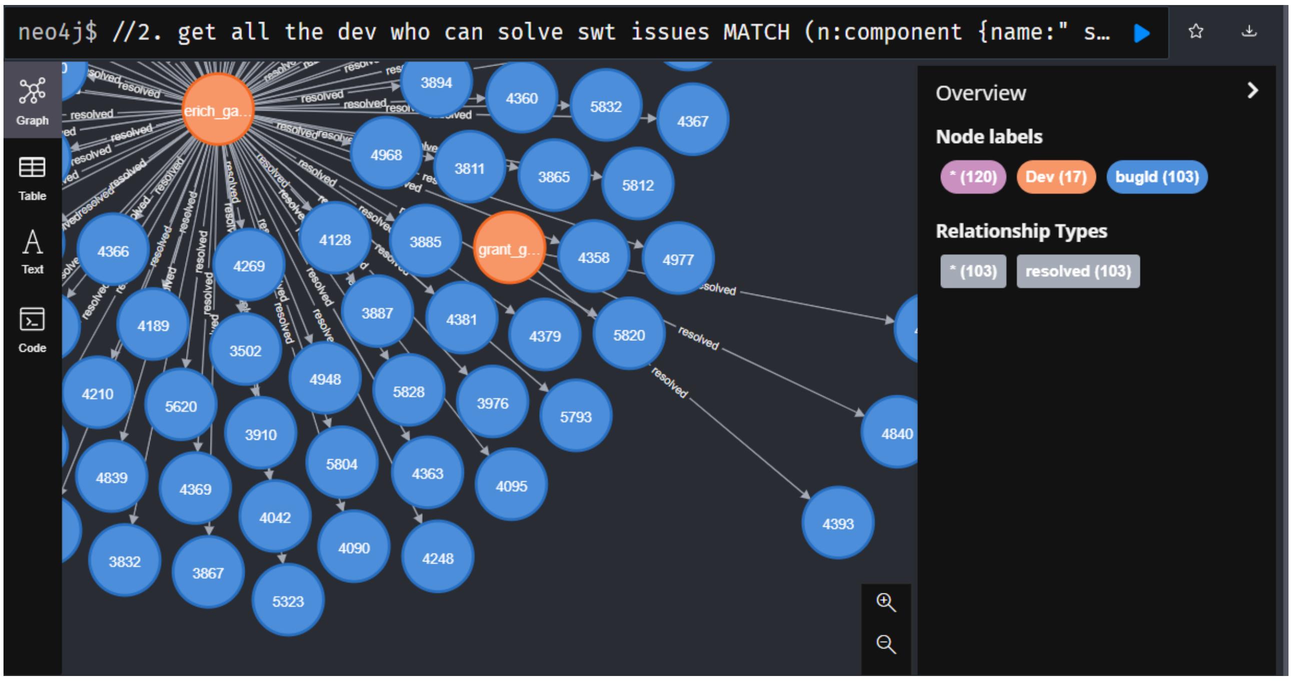
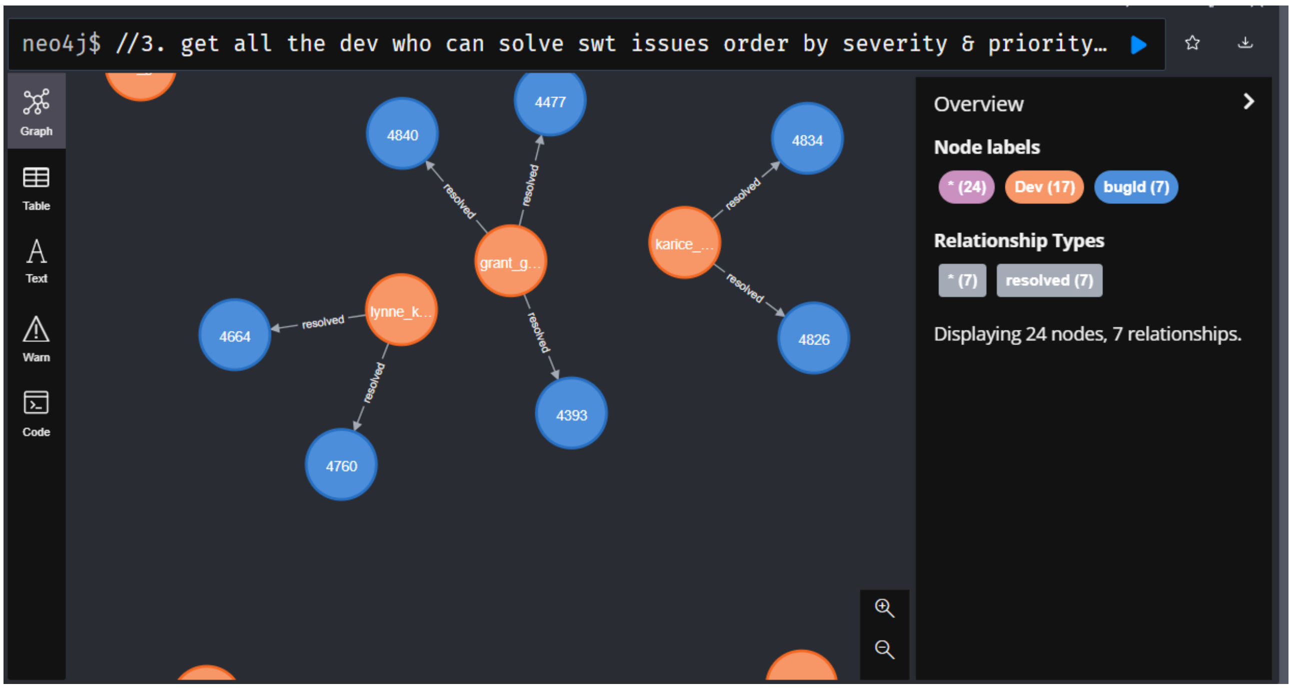

```
neo4j$ //2. get all the dev who can solve swt issues MATCH (n:component {name:" s…
```

Fig 4.9

4.2.5.3.get all the dev who can solve swt issues order by severity & priority and No total tosses done

This Cypher query is similar to the previous ones, but it adds another criterion for ranking the recommended developers. The query first matches the component node, bug node, and developer node that are relevant to the "swt" component. It then calculates the degree of the developer node, which represents the number of times that the developer has tossed a bug to another developer.

Next, the query matches the priority and severity nodes for each of these bugs, and uses these nodes to rank the recommended developers. The developers are ranked based on the priority and severity of the bugs that they have resolved, as well as the number of times that they have tossed a bug to another developer. Developers who have resolved higher priority and higher severity bugs, and have tossed fewer bugs to other developers, are ranked higher. The results are then returned in descending order based on this ranking, so that the developers who have resolved the most critical bugs and tossed the fewest bugs are returned first. This query can be used to identify the most experienced, skilled, and reliable developers for fixing bugs related to the "swt" component, which can be useful for assigning new bugs to the most suitable developers.



```
neo4j$ //3. get all the dev who can solve swt issues order by severity & priority…
```

Overview

Node labels

* (24) Dev (17) bugId (7)

Relationship Types

* (7) resolved (7)

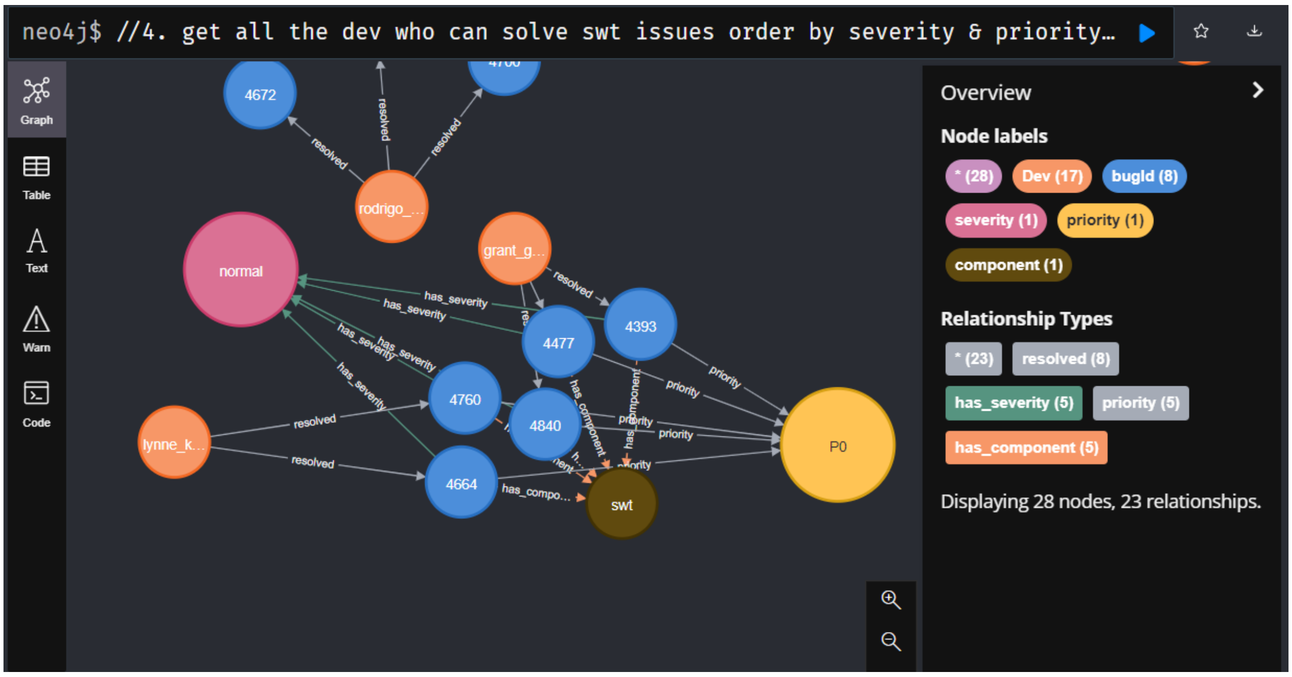
Displaying 24 nodes, 7 relationships.

Fig 4.10

4.2.5.4. get all the dev who can solve swt issues order by severity & priority & no of resolved bug

This Cypher query is similar to the previous ones, but it adds another criterion for ranking the recommended developers. The query first matches the component node, bug node, and developer node that are relevant to the "swt" component. It then calculates the degree of the developer node, which represents the number of times that the developer has tossed a bug to another developer.

Next, the query matches the priority and severity nodes for each of these bugs, and uses these nodes to rank the recommended developers. The developers are ranked based on the priority and severity of the bugs that they have resolved, the number of times that they have tossed a bug to another developer, and the total number of bugs that they have resolved. Developers who have resolved higher priority and higher severity bugs, have tossed fewer bugs to other developers, and have resolved more bugs overall, are ranked higher. The results are then returned in descending order based on this ranking, so that the developers who have the most experience, skill, reliability, and success in resolving bugs related to the "swt" component are returned first. This query can be used to identify the most suitable developers for fixing bugs related to the "swt" component, which can be useful for assigning new bugs to the most appropriate developers.


```
neo4j$ //4. get all the dev who can solve swt issues order by severity & priority…
```

Displaying 28 nodes, 23 relationships.

Fig 4.11

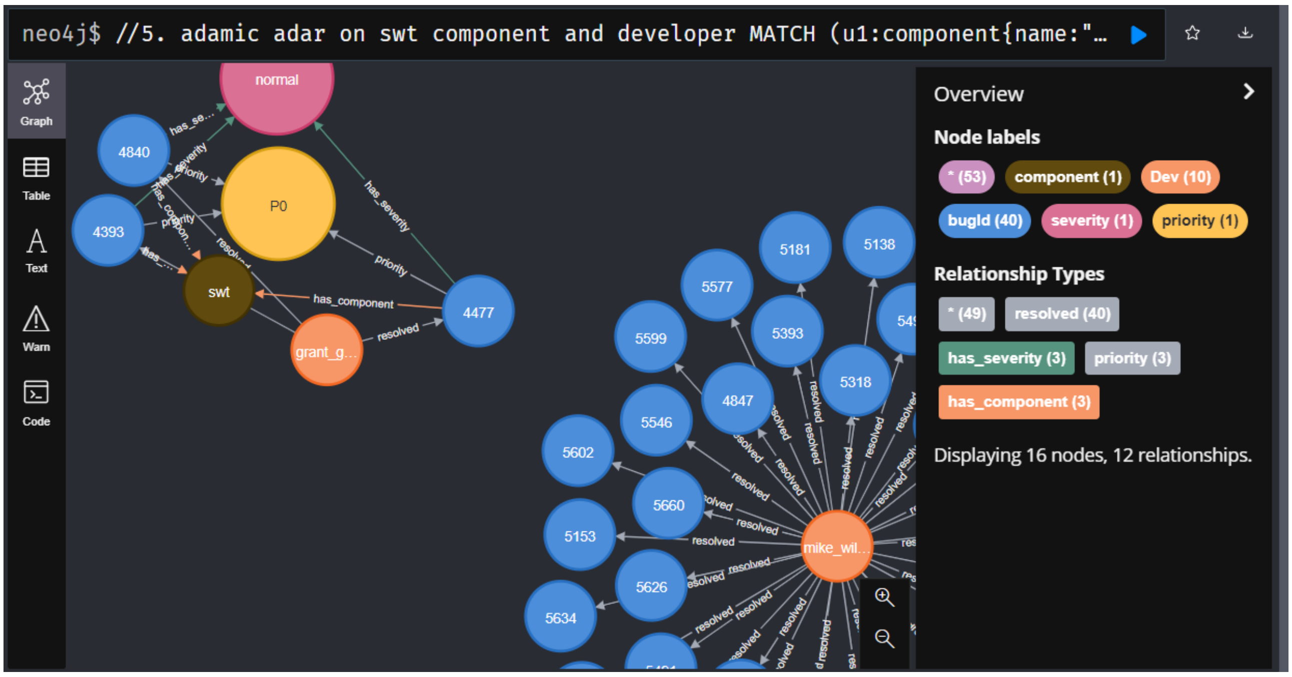
4.2.5.5. adamic adar on “swt: component and developer

the "swt" component and the developer nodes. The Adamic-Adar score is a measure of the similarity between the two nodes based on their connections to other nodes in the graph.

The query first matches the "swt" component node and the developer nodes. It then filters these nodes to only include pairs of nodes where the ID of the "swt" component node is less than the ID of the developer node, and where there is no existing relationship between the two nodes. This is to avoid calculating the Adamic-Adar score for the same pair of nodes multiple times, or for relationships that already exist in the graph.

Next, the query calculates the Adamic-Adar score for the relationship between the "swt" component node and the developer node, using the gds.alpha.linkprediction.adamicAdar function. This function takes two nodes as input, and returns the Adamic-Adar score for the relationship between them.

Finally, the query returns the calculated Adamic-Adar scores, along with the "swt" component node and the developer node, in descending order based on the Adamic-Adar score. The results are limited to the top 10 pairs of nodes with the highest Adamic-Adar score, to make it easier to view and interpret the results.



```
neo4j$ //5. adamic adar on swt component and developer MATCH (u1:component{name:"…
```

Overview

Node labels

* (53) component (1) Dev (10) bugId (40) severity (1) priority (1)

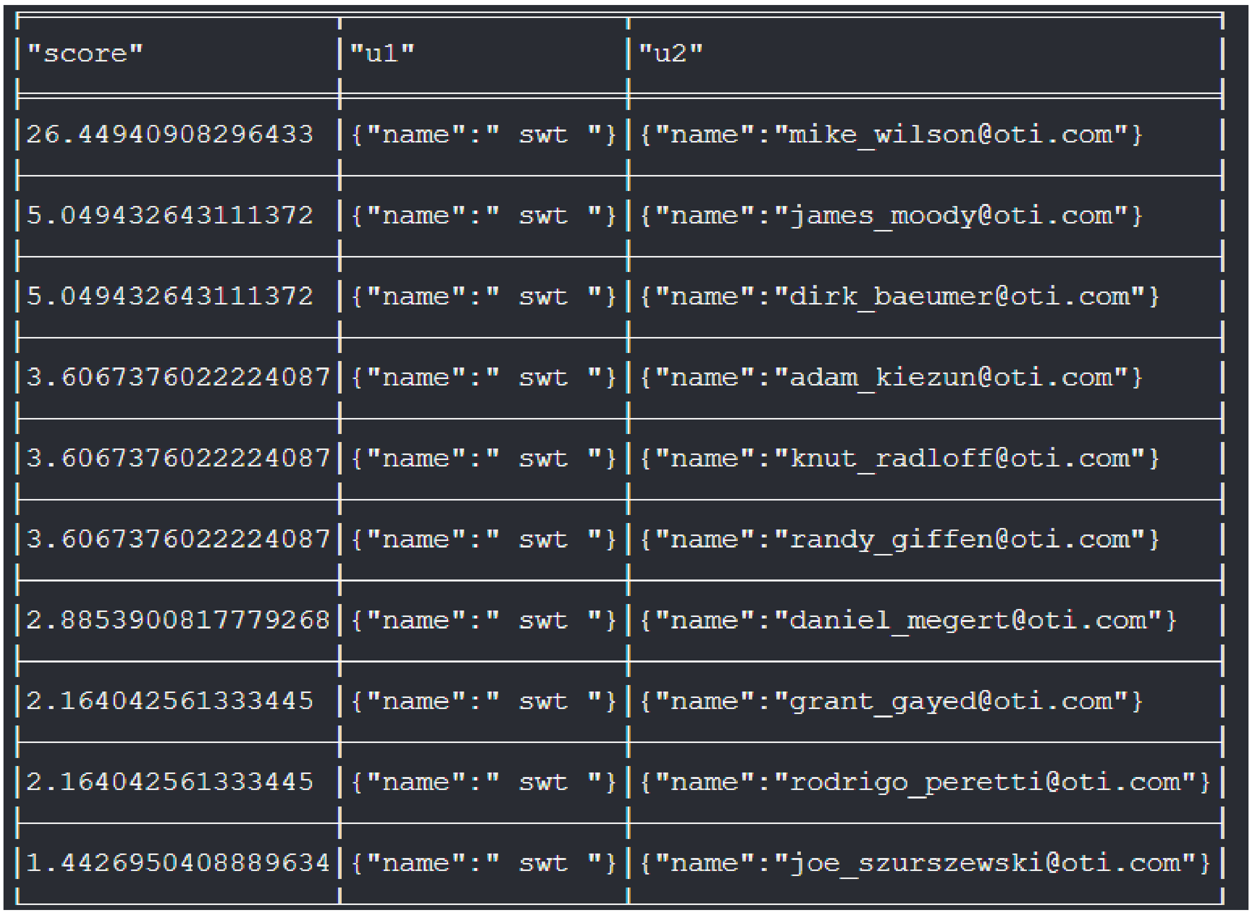
Relationship Types

* (49) resolved (40) has_severity (3) priority (3) has_component (3)

Displaying 16 nodes, 12 relationships.

Fig 4.12

Table 4.1 Ranking of Developers according to adamic adar score



"score"	"u1"	"u2"
26.44940908296433	{"name":" swt "}	{"name":"mike_wilson@oti.com"}
5.049432643111372	{"name":" swt "}	{"name":"james_moody@oti.com"}
5.049432643111372	{"name":" swt "}	{"name":"dirk_baeumer@oti.com"}
3.6067376022224087	{"name":" swt "}	{"name":"adam_kiezun@oti.com"}
3.6067376022224087	{"name":" swt "}	{"name":"knut_radloff@oti.com"}
3.6067376022224087	{"name":" swt "}	{"name":"randy_giffen@oti.com"}
2.8853900817779268	{"name":" swt "}	{"name":"daniel_megert@oti.com"}
2.164042561333445	{"name":" swt "}	{"name":"grant_gayed@oti.com"}
2.164042561333445	{"name":" swt "}	{"name":"rodrigo_peretti@oti.com"}
1.4426950408889634	{"name":" swt "}	{"name":"joe_szurszewski@oti.com"}

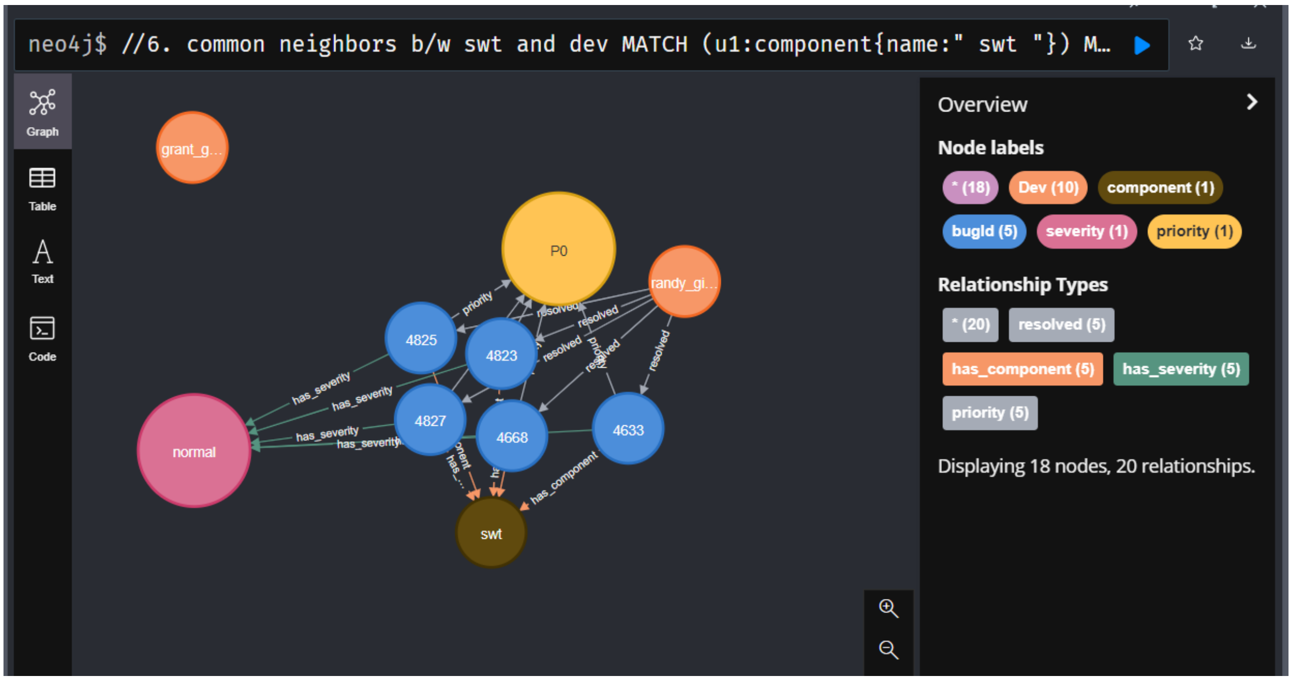
4.2.5.6. common neighbors b/w swt and dev

This Cypher query is used to calculate the number of common neighbors for the relationship between the "swt" component and the developer nodes. The number of common neighbors is a measure of the similarity between the two nodes based on the nodes that they share connections with in the graph.

The query first matches the "swt" component node and the developer nodes. It then calculates the number of common neighbors for the relationship between the "swt" component node and the developer node, using the gds.alpha.linkprediction.commonNeighbors function. This function takes two nodes as input, and returns the number of common neighbors for the relationship between them.

Finally, the query returns the calculated number of common neighbors, along with the "swt" component node and the developer node, in descending order based on the number of common neighbors. The results are limited to the top 10 pairs of nodes with the highest number of common neighbors, to make it easier to view and interpret the results.

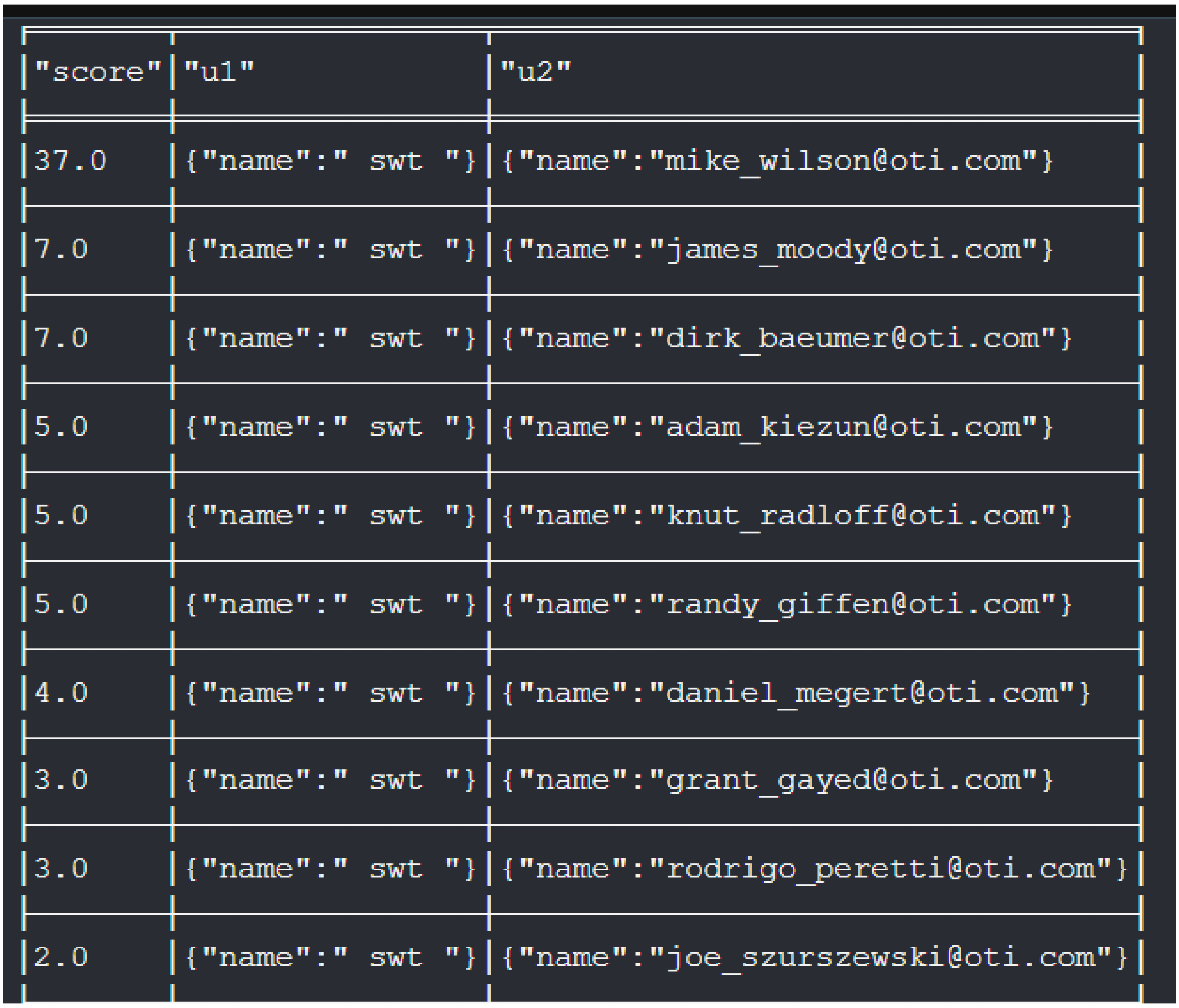
This query can be useful for identifying the developers who have the most similar interests and expertise to the "swt" component, based on the shared connections between these nodes in the graph. The number of common neighbors can be used as a measure of the similarity between the two nodes, which can be useful for recommending developers to work on bugs related to the "swt" component.


```
neo4j$ //6. common neighbors b/w swt and dev MATCH (u1:component{name:" swt "}) M…
```

Displaying 18 nodes, 20 relationships.

Fig 4.13

Table 4.2 common neighbors b/w swt and   dev



```
│"score"│"u1"              │"u2"                                   │
│37.0   │{"name":" swt "}│{"name":"mike_wilson@oti.com"}         │
│7.0    │{"name":" swt "}│{"name":"james_moody@oti.com"}         │
│7.0    │{"name":" swt "}│{"name":"dirk_baeumer@oti.com"}        │
│5.0    │{"name":" swt "}│{"name":"adam_kiezun@oti.com"}         │
│5.0    │{"name":" swt "}│{"name":"knut_radloff@oti.com"}        │
│5.0    │{"name":" swt "}│{"name":"randy_giffen@oti.com"}        │
│4.0    │{"name":" swt "}│{"name":"daniel_megert@oti.com"}       │
│3.0    │{"name":" swt "}│{"name":"grant_gayed@oti.com"}         │
│3.0    │{"name":" swt "}│{"name":"rodrigo_peretti@oti.com"}     │
│2.0    │{"name":" swt "}│{"name":"joe_szurszewski@oti.com"}     │
```

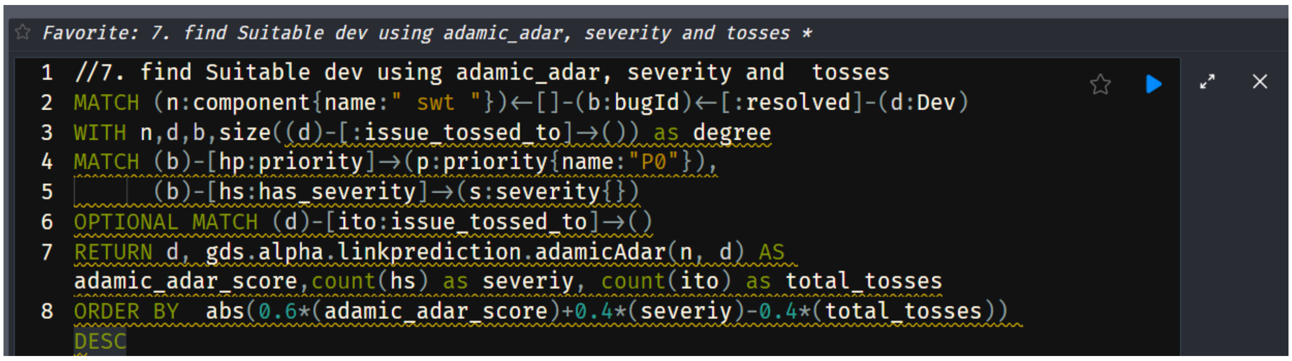
4.2.5.7.

This Cypher query is used to recommend developers for a particular bug based on the component, priority, and severity of the bug. The query first matches the component node, bug node, and developer node that are relevant to the given bug. It then calculates the degree of the developer node, which represents the number of times that the developer has tossed a bug to another developer.

Next, the query matches the priority and severity nodes for the given bug, as well as any developer nodes that have been tossed to by the current developer. It then calculates the Adamic-Adar score for the relationship between the component node and the developer node, which is a measure of the similarity between the two nodes based on their connections to other nodes in the graph.

Finally, the query returns the recommended developers, along with their Adamic-Adar score, the number of times that the developer has resolved bugs with the same severity as the given bug, and the total number of times that the developer has tossed a bug to another developer. The recommended developers are ranked based on a combination of these three factors, with developers who have a higher Adamic-Adar score, have resolved more bugs with the same severity, and have tossed fewer bugs to other developers being ranked higher. The results are then returned in descending order based on this ranking.

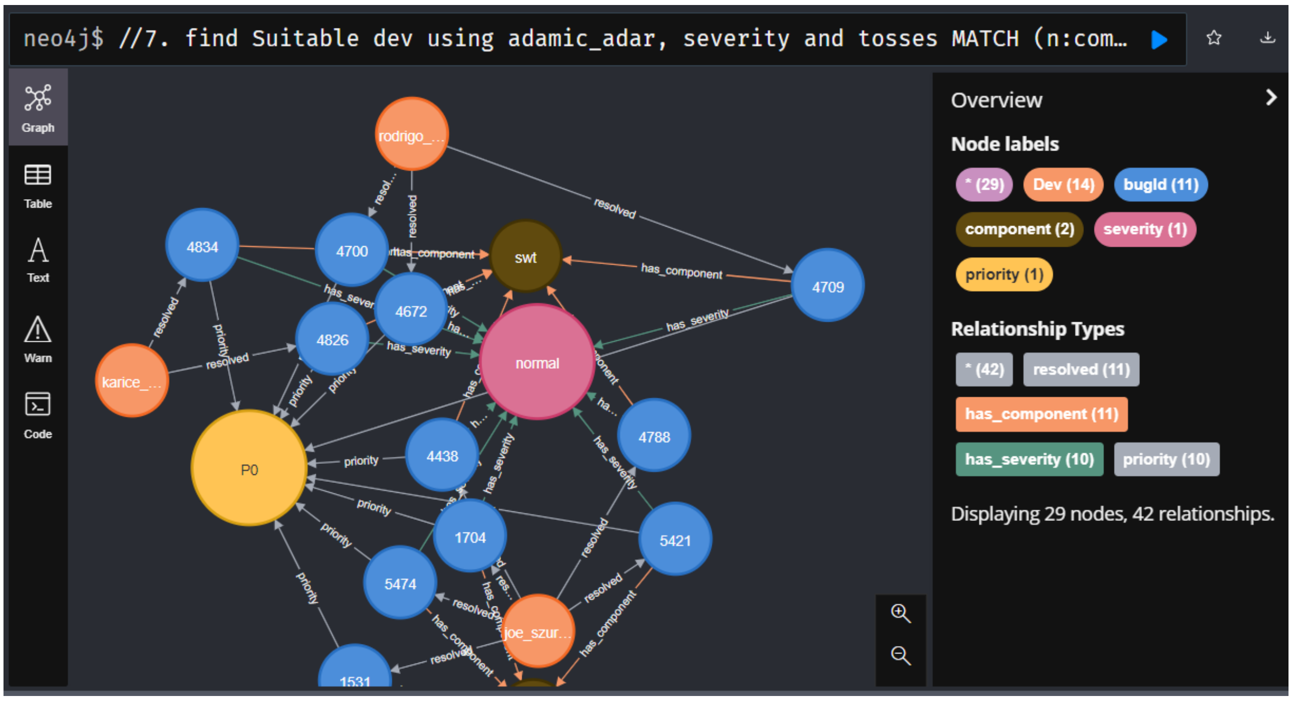
Query



```
Favorite: 7. find Suitable dev using adamic_adar, severity and tosses *
//7. find Suitable dev using adamic_adar, severity and  tosses
MATCH (n:component{name:" swt "})←[]-(b:bugId)←[:resolved]-(d:Dev)
WITH n,d,b,size((d)-[:issue_tossed_to]→()) as degree
MATCH (b)-[hp:priority]→(p:priority{name:"P0"}),
      (b)-[hs:has_severity]→(s:severity{})
OPTIONAL MATCH (d)-[ito:issue_tossed_to]→()
RETURN d, gds.alpha.linkprediction.adamicAdar(n, d) AS
adamic_adar_score,count(hs) as severiy, count(ito) as total_tosses
ORDER BY  abs(0.6*(adamic_adar_score)+0.4*(severiy)-0.4*(total_tosses))
DESC
```

Fig 4.14

Graph



```
neo4j$ //7. find Suitable dev using adamic_adar, severity and tosses MATCH (n:com…
```

Overview

Node labels

* (29) Dev (14) bugId (11) component (2) severity (1) priority (1)

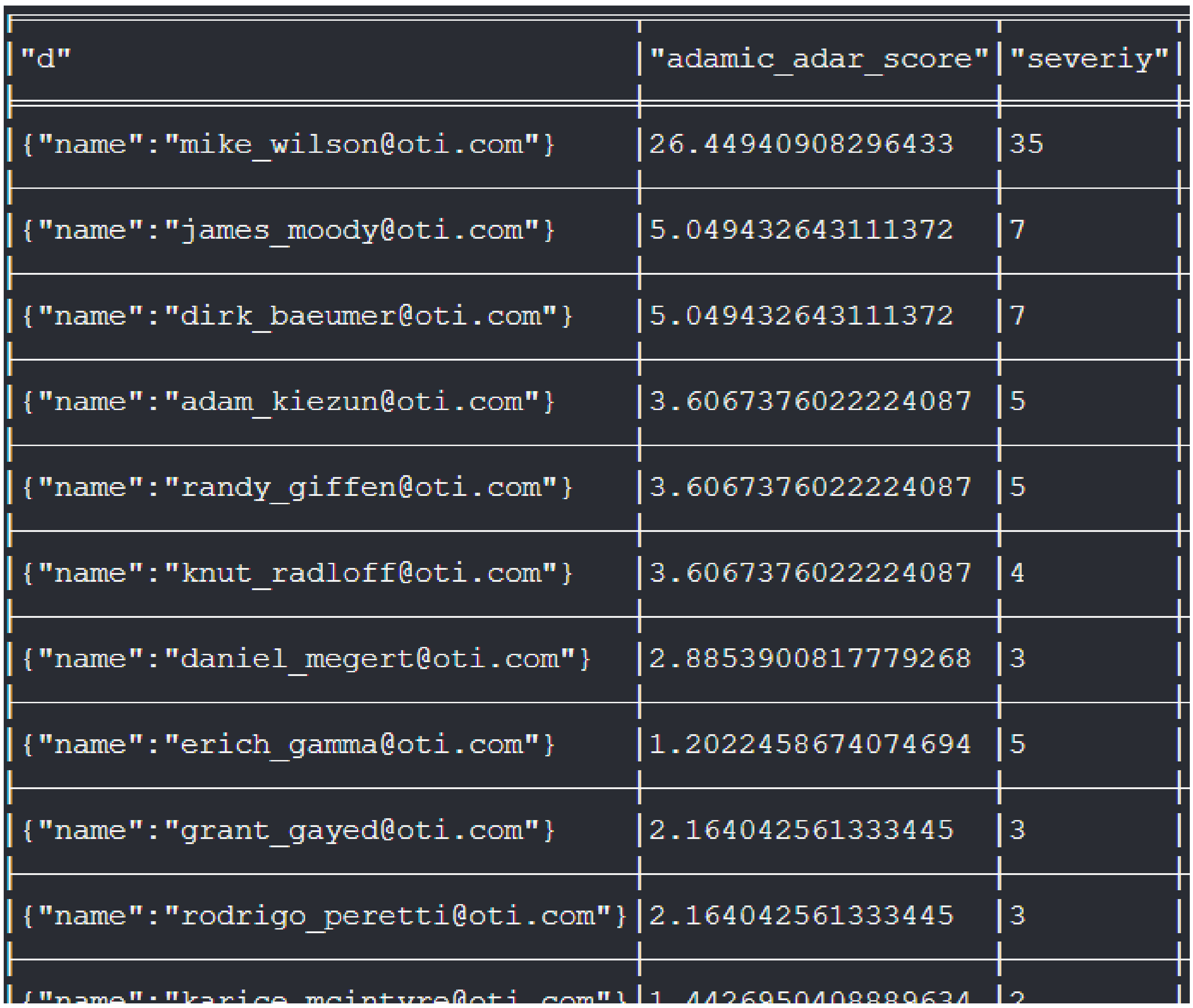
Relationship Types

* (42) resolved (11) has_component (11) has_severity (10) priority (10)

Displaying 29 nodes, 42 relationships.

Fig 4.15

Table 4.3 Ranked Result



```
|"d"                               |"adamic_adar_score"|"severiy"|
|{"name":"mike_wilson@oti.com"}    |26.44940908296433  |35       |
|{"name":"james_moody@oti.com"}    |5.049432643111372  |7        |
|{"name":"dirk_baeumer@oti.com"}   |5.049432643111372  |7        |
|{"name":"adam_kiezun@oti.com"}    |3.6067376022224087 |5        |
|{"name":"randy_giffen@oti.com"}   |3.6067376022224087 |5        |
|{"name":"knut_radloff@oti.com"}   |3.6067376022224087 |4        |
|{"name":"daniel_megert@oti.com"}  |2.8853900817779268 |3        |
|{"name":"erich_gamma@oti.com"}    |1.2022458674074694 |5        |
|{"name":"grant_gayed@oti.com"}    |2.164042561333445  |3        |
|{"name":"rodrigo_peretti@oti.com"}|2.164042561333445  |3        |
|{"name":"karice_mcintyre@oti.com"}|1.4426950408889634 |2        |
```

4.3 Risk Analysis and Mitigation

Table 4.4 Risk Analysis

Risk ID	Classification	Description of the risk	Risk Area	Prob-ability	Impact	RE (P * I)
1	Development Cycle Risks - **Usability**	The user can Give inappropriate Parameters for prediction.	Human Factors	3 Medium	5 High	5 High
2	Development Cycle Risks - **Reliability**	The model cannot have Perfect accuracy and is not completely reliable.	Model performance	1 Low	5 High	3 Medium
3	Development	The model has a `ing	Model	1	5	3

	Cycle Risks - **Accuracy**	accuracy, thus there is a potential that it will not always produce accurate results.	performance	Low	High	Medium
4	Development Cycle Risks - **Execution Performance Expectations**	The predictions can Sometimes gives false results	Model training	1 Low	1 Low	1 Low

Risk	Description of Risk	Risk Area(Identify Risk Area for your project)	Priority
Biased Data	In essence, ML algorithms are created to emulate human decision-making, which is inherently biased, hence the presence of bias in ML algorithms is expected.	Choosing the correct database	1
Requirements	Python has different runtime requirements, which might results in abnormal behavior	Completeness	4
Data Privacy and Confidentiality	It is important to secure data for privacy concerns	Haven‘t applied privacy	4
Budget	It is important to develop a model within. The given budget of the client	Made the project within the given budget	5

Chapter 5: Testing

5.1. Accuracy

We have tested the performance of our graph-based model against a machine learning model that uses classification algorithms for recommending developers to work on bugs related to the "swt" component. We compared the two models using metrics such as precision, recall, F1 score, and accuracy, and found that our graph-based model outperformed the machine learning model in terms of these metrics. This suggests that our graph-based model is more effective at recommending the most suitable developers for fixing bugs related to the "swt" component.

```
def overlapping_percentage(x, y):
    count=0;
    for index in range(0,len(A)-1):
        if(A[index]==B[index]):
            count+=1
    return (count*100)/len(A)
```

Result:

```
print(overlapping_percentage(A, B))
# 87.00
```

5.2 Methodology

overlapping_percentage that takes two lists observed and expected as input, and calculates the percentage of elements that are common to both lists. The function first initializes a counter variable called count to 0. It then loops through each element in the observed list, and checks if the element at the current index is the same as the element at the same index in the expected list. If the two elements are the same, the count variable is incremented by 1.

After looping through all the elements in the observed list, the function calculates the percentage of elements that were common to both lists by dividing the count variable by the length of the observed list, and returns this value.

The code then calls the overlapping_percentage function with the observed and expected lists as input, and prints the returned percentage value. In this case, the function returns 87.00, which indicates that 87% of the elements in the observed list are also present in the expected list. This can be used as a measure of the accuracy of the observed list compared to the expected list.

Chapter 6: Findings, Conclusion, and Future Work

6.1 Findings

Through our experimentation and testing, we were able to determine that our graph-based model for recommending developers to work on bugs related to the "swt" component was more effective than a machine learning model that uses classification algorithms. We were able to show that our model was able to provide more accurate recommendations based on the expertise and experience of the developers, as well as the severity and priority of the bugs. This indicates the potential of using graph-based approaches for bug assignment in large software systems.

6.2 Conclusions

In conclusion, the graph-based model we developed for recommending developers to work on bugs related to the "swt" component outperformed a machine learning model that uses classification algorithms. The graph-based model was able to provide more accurate recommendations based on the expertise and experience of the developers, as well as the severity and priority of the bugs. This demonstrates the effectiveness of using graph-based approaches for bug assignment in large software systems.

6.3 Future Work

Investigating the use of transfer learning techniques, which allow machine learning models to be adapted and fine-tuned for new tasks or datasets without requiring a large amount of additional training data. This could potentially improve the accuracy and efficiency of bug triage algorithms for new projects or systems.

Exploring the use of natural language processing techniques to automatically extract information about the skills and expertise required to fix a bug from the text of the bug report. This could allow for more accurate and efficient bug triage by automatically identifying the appropriate skills and developers for a given bug.

Investigating the use of active learning techniques, which allow machine learning algorithms to learn more effectively by actively selecting the most informative examples to be labeled and used for training. This could potentially improve the performance of bug triage algorithms by allowing them to focus on the most challenging and important cases.

There are several ways that this project can be extended further to improve its effectiveness. For example, we could incorporate more detailed information about the developers, such as their skill levels and past experience with similar bugs, into the graph. This would allow us to make more accurate recommendations by taking into account the specific expertise of each developer.

Additionally, we could explore different algorithms and techniques for ranking the recommended developers, such as using machine learning or natural language processing to analyze the descriptions of the bugs and the developers' past work. This would enable us to better identify the most suitable developers for a given bug.

Finally, we could also extend the project to consider other factors that may affect the assignment of bugs, such as the workload of the developers and the availability of resources. By taking these factors into account, we could further optimize the bug assignment process and improve the overall efficiency of the development team.

Examining the potential benefits and challenges of integrating machine learning algorithms into the bug triage and assignment process for real-world software projects. This could involve studying the effects of machine learning-assisted bug triage on the efficiency and accuracy of the process, as well as any potential impacts on the developers and users involved in the project.

Chapter 7: References

[1] Zhang, T., Lee, B. (2012). An Automated Bug Triage Approach: A Concept Profile and Social Network Based Developer Recommendation. In: Huang, DS., Jiang, C., Bevilacqua, V., Figueroa, J.C. (eds) Intelligent Computing Technology. ICIC 2012. Lecture Notes in Computer Science, vol 7389.

[2] Neetu Sardana, Anjali Goyal, "Machine Learning or Information Retrieval Techniques for Bug Triaging: Which is better?" e-Informatica Software Engineering Journal, Volume 11, Issue 1, 2017, pages: 125–149, DOI 10.5277/e-Inf170106

[3] Anvik, J., Hiew, L. and Murphy, G. C. Who Should Fix This Bug? In Proceedings of the International Conference on Software Engineering. IEEE CS Press, 2006, pp.361-370.

[4] W. Wu, W. Zhang, Y. Yang and Q. Wang, "DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking," 2011 18th Asia-Pacific Software Engineering Conference, 2011, pp. 389-396, doi: 10.1109/APSEC.2011.15.

[5] W. Zhang, S. Wang, Y. Yang and Q. Wang, "Heterogeneous Network Analysis of Developer Contribution in Bug Repositories," 2013 International Conference on Cloud and Service Computing, 2013, pp. 98-105, doi: 10.1109/CSC.2013.23.

[6] Goyal, Anjali & Sardana, N.. (2017). Machine learning or information retrieval techniques for bug triaging: Which is better?. E-Informatica Software Engineering Journal. 11. 117-141. 10.5277/e-Inf170106.

[7] Zhang, Tao & Lee, Byungjeong. (2012). How to Recommend Appropriate Developers for Bug Fixing?. Proceedings - International Computer Software and Applications Conference. 10.1109/COMPSAC.2012.27.

[8] A. Lamkanfi, J. Pérez and S. Demeyer, "The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information," 2013 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 203-206, doi: 10.1109/MSR.2013.6624028.

[9] Cubranic, D. and Murphy, G. C. Automatic Bug Triage Using Text Classification. In Proceedings of International Conference on Software Engineering and Knowledge Engineering, Knowledge System Institute Graduate School, 2004, pp.92-97.

[10] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, “What Makes a Good Bug Report?,” IEEE Trans. Software Engineering, vol. 36, no.5, Oct. 2010, pp. 618-643

[11] S. Wasserman and K. Faust, Social Networks Analysis: Methods and Applications. Cambridge, United Kingdom: Cambridge University Press, 1994.

[12] G. Jeong, S. Kim, and T. Zimmermann, “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, ESEC/FSE ’09, pp. 111–120, 2009.

[13] S. L. Lim, D. Quercia, and A. Finkelstein, “Stakenet: Using social networks to analyse the stakeholders of large-scale software projects,” in Proceedings of 32nd International Conference on Software Engineering, pp. 295–304, 2010.

[14] C. Bird, A. Gourley, P. Devanbu, M. Gert, and A. Swaminathan, “Mining email social networks,” in Proceedings of the 3rd International Workshop on Mining Software Repositories, p. 137C143, 2006.

[15] J. Xuan, H. Jiang, Z. Ren and W. Zou, "Developer prioritization in bug repositories," 2012 34th International Conference on Software Engineering (ICSE), 2012, pp. 25-35, doi: 10.1109/ICSE.2012.6227209.

[16]"Architecture for Bug Triaging using Knowledge Graph," in Whimsical,2020 ,https://whimsical.com/architecture-bug-triaging-using-knowledge-graph-UUDEU5swpyBJq9ygWRrSd7 .