UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Software Engineering Curriculum

İlgün İlgün

Tools for software project data collection and integration

Master's Thesis (30 ECTS)

Supervisor(s): Siim Karus

Tartu 2015

Tools for software project data collection and integration

Abstract:

Nowadays Social Media has a big impact in our society and also in the software development process. Everyday more and more people are communicating through social media and discussing about their life and even for their work processes. Unlike the past century it is much easier to integrate teams even if they have oceans between them. Tools such as JIRA, TFS and Bugzilla are created for that purpose: Integrating teams and making life easier for everyone who is taking part of any cycle of software development process. This Master's Thesis aims to integrate social media with issue trackers and analysing the relationships between them. In this thesis, unified models for social media and issue trackers were designed. Reverse engineering is used for designing the models and unification of the data models. After, creating the unified models, adapters were written in order to extract the data from social media and issue trackers for analysing. We conducted an example analysis on the data that we got by merging issue tracking. We found out that some interesting facts such as, open source software projects' contributors are tend to communicate via IRC and email lists and also 76% of the users who are active in IRC are also active in issue tracking systems.

Keywords:

Issue tracking systems, Social media, JIRA, Bugzilla, IRC, Twitter, Mailing Lists

Tööriistad tarkvaraprojekti andmete kogumiseks ja integreerimiseks

Lühikokkuvõte:

Sotsiaalmeedial on tänapäeval suur roll ühiskonnas ja tarkvaraarendusprotsessis. Iga päevaga kasvab sotsiaalmeedia vahendusel suhtlevate ja oma elu ja tööprotsesse kajastavate inimeste arv. Erinevalt möödunud sajandist on palju lihtsam integreerida meeskondasid isegi siis kui neid lahutab ookean. Tööriistad nagu JIRA, TFS ja Bugzilla on loodud just sel eesmärgil: meeskondade integreerimine ja tarkvaraarenduse protsessis osalejate elu lihtsustamine. Selle magistritöö eesmärk on siduda sotsiaalmeedia muutustehaldusega ning analüüsida nende vahelisi seoseid. Selles töös loodi ühine mudel muutustehalduse ja sotsiaalmeedia jaoks. Mudelite loomiseks ja ühendamiseks kasutati pöördprojekteerimist. Pärast ühise mudeli loomist, kirjutati adapterid sotsiaalmeediast ja muudatustehaldusest andmete ühisesse mudelisse laadimiseks. Muudatustehalduse ja sotsiaalmeedia kanalite andmete ühendamisega saadud andmestikul teostati näidisanalüüs. Analüüs näitas, et avatud lähtekoodiga tarkvaraprojektidesse panustajad suhtlevad IRC ja e-maili listide teel ning 76% IRC kasutajatest on ka aktiivsed muudatustehalduse kasutajad.

Võtmesõnad:

Muudatustehaldussüsteemid, Sotsiaalmeedia, JIRA, Bugzilla, IRC, Twitter, Mailing Lists

Table of Contents

1 Introduction

There was a paper which is interested on the community structures of open source software projects [1]. Issue trackers also known as bug trackers are basically repositories which can keep track of your issues such as bugs, refactoring issues, new features or production issues. These systems help your software development speed significantly and most importantly they provide internal communication for everyone who is contributing to the project. One of the most popular issue tracker system especially in Open Source community is Bugzilla which is an open source issue tracker used in many projects.

On the other hand, we have social media where community of open source software world communicates since the contributors to these projects are mostly volunteers. Thus, these volunteers can be anywhere in the world and generally are not met face-to-face. Thus, issue trackers and social media are two separate things and no communication or integration between them are readily available. Integration between these is relevant because there can be questions asked such as "Are Twitter users also active in issue tracking systems?" For integrating them we used unified models which were designed for both types of communication. One unified model was designed for social media channels which is essential for extracting data from them. Other one was designed for issue trackers.

This thesis is divided into seven chapters. Chapter 2 provides background information regarding the thesis such as open source software world is as well as social media and issue tracking systems. In the background on Chapter 3 methodology and data sources are discussed including how we parsed and stored the data which comes from the case studies. In Chapter 4, creation of the unified models is explained. In Chapter 5, implementation of adapters which are created for collecting the data from case studies explained. In Chapter 6, our proof of concept queries can be found as well as a discussion about their results. In Chapter 7, we discuss related works. In Chapter 8, conclusion and ideas for the future work can be found.

1.1 Contribution

In this section, we listed our main contributions as well as the proof of concept queries, proof of concepts queries are example analysis that we made in order to proof our study is valid.

Our main contributions are:

1. Unified model created for social media channels.
2. Unified model created for issue tracking systems.
3. Adapters created in order to get data for analysis.

In order to show some of the benefits of a unified data model, we make some proof of concept queries on the model. We have picked these queries because we simply thought that they were facts to found out the answers through our case studies. Our proof-of-concept queries are as follows:

a) If users in IRC change nicks in Hibernate, are they actively using JIRA?

b) Which social media channel is more actively used in Hibernate?

c) If a user is active in IRC, then is he also active in JIRA?

d) If a user changes nick in IRC, then is he reporter or assigner in JIRA?

e) If a user replies to email archives, is he reporter or assigner?

f) If a user is active in email archives then, is he active also in issue tracking systems?
g) If a user is active in Twitter then, is he active also in issue tracking systems?
h) If user joins more than 10 times in IRC, if he active in JIRA?
i) If user joins more than 10 times in IRC, is he more likely to be a reporter or assignee in JIRA?
j) If user quits more than 20 times in IRC, is he active in JIRA?
k) If user quits more than 20 times in IRC, is he more likely to be reporter or assignee in JIRA?

2 Background

One who reads the paper has to know about the concepts such as open source software and why the contributors of open source software world have to communicate through social media channels. Also the reader should be aware of what social media is and which of them are used in the paper as well as an overall idea of how they work.

2.1 Open Source Software

In open source project world, the source code of a computer program is publicly available and usually shared via the Internet. Every sufficient skilled internet user can reach this code and directly start contributing to the project at any time. Contribution can be made in different aspects such as fixing a bug and/or implementing a feature. Although there are some core developer for some projects, open source project are developed "for free", generally as a hobby. However, some companies are sponsoring the open source software development, and pay developer so that they can work on the project as full-time. In the beginning, it seems like that changes the behaviour of open source software development, but the general principles remain same that source code is available to everybody.

Some major concepts of open source software development are as follow [2]:

- Distributed software
- Free software
- Available source code
- Communicate through internet
- Developers are users
- Unpaid and large amount volunteers

Table 1. Pros of open source development model [3].

	For Users	For Developers	For System
Pros	1. Flexibility 2. Strong value 3.Participant in interested part 4.Code availability 5.Ability to modify the code 6.Knowledge sharing 7.Increasing motivation 8.Greater choice and control	1. Allow to make own solution. 2. Reuse many existing functionalities. 3. Allow to survey problems freely. 4. Reduce damages in the beginning time of the system 5. More motivation	1. Bug Detecting 2. OS tools 3. Reliability 4. Customizability 5. Sophisticated 6. Cost effective 7. Rapid evolution 8. Portability 9. Extensibility 10. Reusability 11. Little cost 12. Multitude licensing

Table 2. Cons of open source development model [3].

	For Users	For Developers	For System
Cons	1.Useless documentation 2.Unstructured development 3.Irresponsible individuals	1. Lack of tools 2. Collaboration with new developers 3. Review of large projects	1. Lack of formal process; centralized management release and documentation. 2. Poor design 3. Hard estimation of man power 4. No single responsibility for problem, lack of liability 5. Version proliferation 6. Complex licenses 7. High short term cost

Open source software is kind of a distributed software development that has large numbers of contributors. These contributors have to communicate with each other since most of them do not have the chance to meet in person. This communication can be done in many channels such as IRC, emails, and most recently Twitter in general Social Media.

2.2 Social Media

Since the developers of open source projects are distributed across the world, they rely on social media to communicate with each other. Therefore we have a unified model of social media channels for extracting the data from them in a structured format. We have extracted information from well-known social media channels such as IRC, Twitter and mailing archives.

IRC

More recently, developer are using IRC channels for holding meetings. IRC (Internet Relay Chat) has been around since late 1980s. However, the use of IRC channels is recently rising, some years ago neither Apache nor Mozilla had IRC channels, and today they both have [4]. These meeting can be thought as physical team meeting which you discuss with your team and get direct answers to your questions. More or less IRC communication can be thought as a supportive way for emailing lists where you want to have a quick discussion with your team. IRC was created by Jarkko Oikarinen in august 1988 to replace a program called MUT (MultiUser Talk) due to bad habits of MUT[1]. The IRC protocol has been designed for text based conferencing which is based on the client-server model, and is well suited to running on many machines in a distributed way. A single server forms the simplest IRC network[2]. IRC has a line-based structure with the client sending single-lined message to the server, receiving replies on these messages and receiving copies of some messages sent by other users. Client are also able to enter some commands by prefixing a '/'[3].

[1] http://www.mirc.com/jarkko.html
[2] https://tools.ietf.org/html/rfc1459
[3] https://tools.ietf.org/html/rfc1459#section-2.3.1

Some useful commands can be listed as[4]:

- JOIN is for joining a channel
- PART is for removing thee from the active users list
- MODE is for changing mode of nicknames and channels
- TOPIC is for changing the topic in a given channel
- NAMES is for listing the nicknames that are visible to them in any channel
- LIST is for listing channels and their topics
- INVITE is for inviting users to a channel.
- KICK is for removing a user from a given channel.

Mailing Lists

A mailing list can be thought as a forum which not only developers, also project manager, bug reporters, users in other words everyone who is involved in software development process contributes to. These lists can be used for exchanging ideas, reporting bugs and finding solutions. Mailing lists started with the invention of internet, it is a fast way of communication for large group of people. A reflector will send a copy of message to all subscribers in the mailing lists. This reflector is also a single email address which is used to distribute all the communication to the subscribers. Mailing lists can be used as announcement lists or/and discussion lists. In announcement lists, only selected people can post however, in discussion lists everyone is free to post whenever they want. In software world, it is more used as a discussion lists which everyone can post.

Twitter

Twitter is a social networking service that allows users to send message up to 140 characters so called tweets. Twitter has 288 million monthly active users with 500 million tweets sent per day [5]. Users can access Twitter using their mobile phones, web interface or SMS. Twitter Inc. which is based in San Francisco has 3600 employees in offices around the world and 50% of them are engineers [5]. Open source developers are using Twitter as well as projects generally have a Twitter account. Developer can communicate through Twitter, ask questions to each other or to the project's account directly.

2.3 Issue Trackers

Nowadays, in software development industry issue trackers are having such a big role. These tools are incredibly useful for having "to-do" items as well as the communication and attachments between users. Using an issue tracker which can also be named as a bug tracker helps significantly team members to communicate and coordinate. These tools are automatically notifying related users with the updates as well. What we care about the issue trackers is the communication between the users and the effect of it to the project. Bugzilla, JIRA and TFS are the most popular ones in the industry, thus we are extracting data from these issue trackers for our study. The issue tracker just don't helps software teams manage issue reporting, assignment, tracking, resolution, and archiving, they also serves as an archive of completed work [6]. The textbook definition of the issue tracking is:

Issue tracking, often called bug tracking (and sometimes request tracking), is the process of keeping track of your open development issues. Bug tracking is a misleading term in many ways and obviously depends on your definition of bug. "Issue" is a broad enough

[4] https://tools.ietf.org/html/rfc1459#section-4.2.1

term to describe most of the kinds of tasks you might need to track when developing [*software*]*, and so drives our choice of terminology here* [7].

And for more technical description such as this:

A bug tracking system is some program or application that allows the project team to report, manage, and analyze bug reports and bug trends. Functionally, most bug tracking systems provide a form that allows us to report and manage specific bugs, a set of stored reports and graphs that allow us to analyze, manipulate, and output this bug data in various ways, and a customizable workflow or life cycle that provides for orderly bug management [8].

Bugzilla

Bugzilla is currently an open-source web-based issue tracker originally developed by Mozilla project. After being released as an open-source project, Bugzilla has been used by most of the open-source projects as their issue tracking tool due to free license of Bugzilla.

Bugzilla firstly came online in 1998. Bugzilla was originally written by Terry Weissman for use at Mozilla.org to replace the in-house solution used by Netscape at that time. Initially Bugzilla was written in TCL. Before the release as an open-source project, Terry decided to port it over to Perl hoping that since Perl seemed to be a popular language thus, it would attract more attention from developers[5].

Bugzilla can be used for:[6]

- Tracking bugs and code changes
- Communicate with teammates
- Submit and review patches
- Manage quality assurance
- Systems administration
- Deployment management
- Chip design and development problem tracking
- Software and hardware bug tracking
- IT support queues

JIRA

JIRA is another issue tracking tool which is developed by Atlassian. Unlike Bugzilla, JIRA is not free unless the organization that will use JIRA is a non-profit one. JIRA has been developed since 2012. JIRA is written in Java and uses Pico inversion of control container, Apache OFBiz entity engine, and WebWork 1 stack technology. For remote procedure calls (RPC), JIRA supports REST, SOAP, and XML-RPC. JIRA integrates with source control programs such as Clearcase, CSV, GIT, Mercurial, Perforce, Subversion and Team Foundation Server[7]. According to Atlassian, JIRA is used for issue tracking and project management by over 40.000 customers[8]. Some of the organization uses JIRA for bug-tracking and project management are Square, Ebay, NASA, Cisco, Salesforce, Adobe, LinkedIn, and BNP Paribas[9].

[5] http://www.bugzilla.org/status/roadmap.html
[6] http://www.bugzilla.org/about/
[7] https://confluence.atlassian.com/display/JIRA/Integrating+JIRA+with+Code+Development+Tools
[8] https://www.atlassian.com/company/customers
[9] https://www.atlassian.com/software/jira

JIRA can be used for: [9]

- Project tracking
- Tracking bugs and code changes
- Manage quality assurance
- Communicate with teammates

2.4 The Lifecycle of an issue

In an issue tracking system, each issue (or bug, or item) generally follows a path from the time that it's created and to the time it's resolved (or closed). This sets of steps (or statuses, or states) often called as "workflow" supported by an issue tracking system [9, 10]. Minimal set of states for an issue should be at least an "open" state and "closed" state. Whenever a request is created it starts with an "open" state and after the work for the issue it eventually gets into "resolved" transaction and then, state should be "closed". If issue later found to be incomplete (or not fixed properly or bug found) it can follow transaction "reopen" and then back to the "open" state.

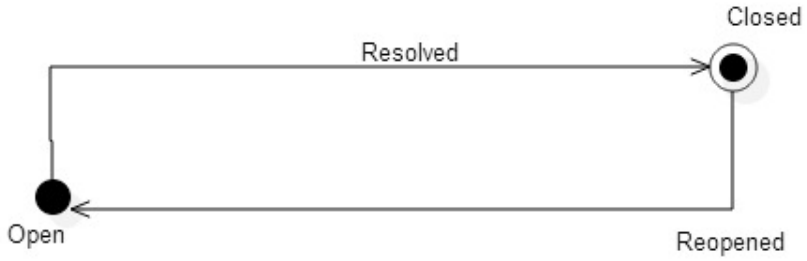

Figure 1. Simplified issue lifecycle.

One can see from the diagrams of Bugzilla (see Figure 2) and JIRA (see Figure 3) issue workflows, there are much more complex examples in real life. In addition, there can be custom states defined by users of issue tracking tools. By customizing issue states, developers can get more idea about the exact state of issues.

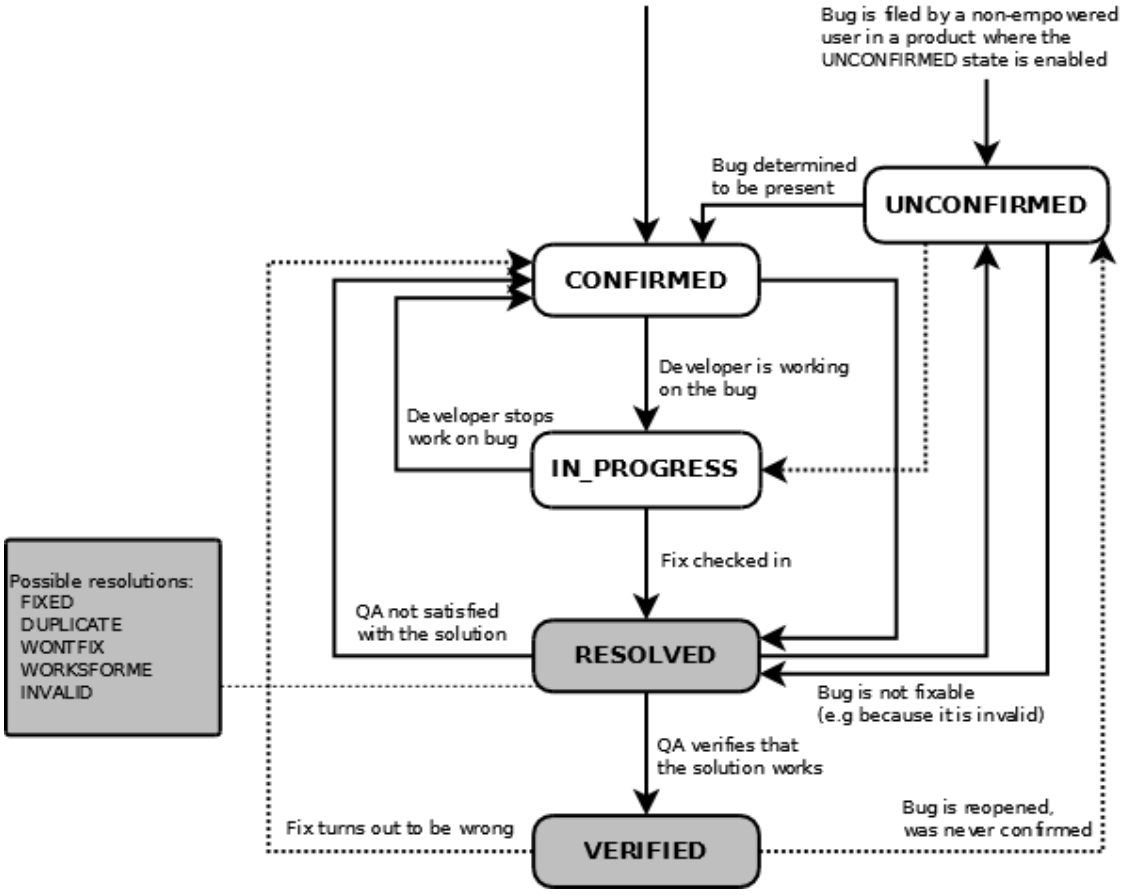

Figure 2. Issue lifecycle supported by Bugzilla [9].

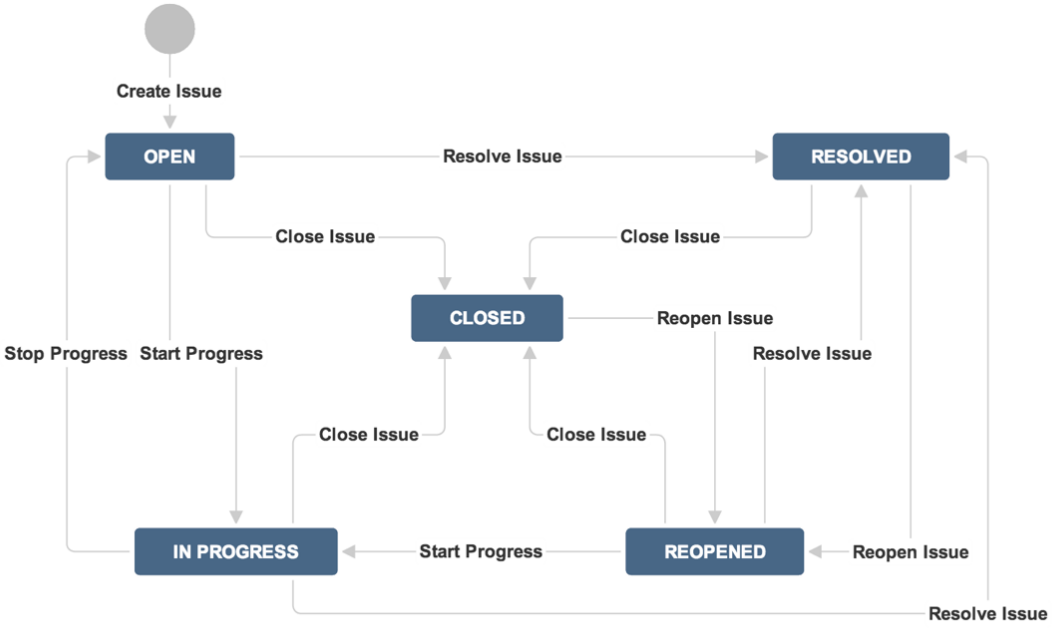

Figure 3. Issue lifecycle supported by JIRA [10].

3 Methodology and Data source

This section describes case studies that we chose and how we collected, parsed, and finally stored these data from case study projects. We chose open source projects as our case studies. Especially Hive and Hibernate are well-known long living open source projects.

3.1 Case Studies

We had chosen to study project Hive, Pulp and Hibernate because there are big projects for extracting data and most importantly they are developed by the developer who are volunteers. They rarely meet in person, so that all the communication have to be done in social media such as mailing lists, IRC and Twitter.

Hive

Hive is one of the apache projects which provides data warehouse software facilities such as querying and managing large datasets residing in distributed storage. Hive is built on top of apache Hadoop. One of the best functions of hive is providing query execution via MapReduce. Hive has its own file system called HDFS (Hadoop distributed file system). Some other features of hive are:[10]

- Tools to enable easy data extract/transform/load.
- A mechanism to impose structure on a variety of data formats
- Access to files stored either directly in HDFS or in other data storage systems such as Apache HBase.

Hive also provides its own query language called QL which is similar to SQL. There is a possibility of writing your own User defines functions. Hive offers scalability which means scaling out with more machines and adding them dynamically to Hadoop[10]. Hive is written in Java and licensed under GNU general public license. Hive is being developed by contributors.

Pulp

Pulp is used for managing repositories of content, such as software packages and publishing those packages to consumers. You can host your content with pulp and manage it easily[11].

Pulp has both server and consumer architecture. With pulp server, one can pull content to pulp server, uploading new content to pulp server, publish content as a web-based repository. With consumer side, one can register into a pulp server and have installed content managed from repository. Pulp support rpm package types and puppet modules. Pulp is generally used for walking software packages through testing, development and stable repositories and pushing those updates out to client packages as they get promoted[11].

Hibernate

Hibernate is an object relational mapping library for Java which provides a framework for mapping an object-oriented-domain model to a traditional relational database[12]. Hibernate provides high level object handling function which solves the problem of mismatches.

[10] https://cwiki.apache.org/confluence/display/Hive/Home
[11] http://www.pulpproject.org/
[12] http://en.wikipedia.org/wiki/Hibernate_(Java)

Hibernate is a free software that is distributed under the GNU Lesser General Public Licence[12].

Hibernate's primary feature is to map from Java classes to database tables and from Java types to database types. Hibernate also provides data query tools. It automatically generates SQL calls and retrieves data for the user. Hibernate implements JPA (Java persistence API) specification. Thus, it can be used from any environment such as Java EE or Java SE that supports JPA[13].

3.2 Data Collection and Parsing

This section describes how we collected data from the external data sources for both social media channels and issue tracking systems. In addition, how we parsed these data into our unified models.

Issue Tracking Repositories

We have two well-known issue tracking repositories in our study JIRA and Bugzilla. Both of them have their hard parts and easy parts. As an example, using JIRA's REST API is much easier and powerful than using Bugzilla's REST API.

JIRA

In order to ingest JIRA issue repositories, we have used JIRA REST API of JIRA. JIRA have many endpoints that we can use for getting raw Json[14]. After getting initial Json from the server for our specific query for a given project, then, we are able to get the total number of issues. This number is used in the loop for getting all the issues for a given project. Although, JIRA REST API is powerful, we have needed more than that. Especially for getting the history of a given issue. JIRA also offers an option for these information by simply extending the search query by changelog. An example query should be something like

```
/rest/api/latest/search?jql&startAt='startCount'&maxResults=
100&expand=changelog
```

In order to build a universal system which would work for many JIRA repository, we implemented a function which is using the REST endpoint of JIRA. However, this endpoint is not present in some versions of JIRA. Thus, this would not be a universal solution for all JIRA repositories. After this step, all the necessary information for a given issue is in our hands. The rest is parsing Json and storing the parsed Json in the database.

Bugzilla

Ingesting Bugzilla issue repositories are harder than JIRA because most of the open-source projects' bugzilla versions are not supporting the REST API of bugzilla. In other words they are pretty old, thus they don't have rest support. In this case, we used a Java library which is basically a wrapper around XML-RPC support of bugzilla[15]. This library is quite useful and supports almost all the needs of this project, however, it doesn't support getting history information. Thus, we implemented a logic which pulls the issue history from the Bugzilla repository directly. This function gets the history as HTML and then Application parses html text into Java objects. In that case, the issue entry is saved into the database.

[13] http://hibernate.org/orm/
[14] https://docs.atlassian.com/jira/REST/latest/
[15] http://techblog.ralph-schuster.eu/b4j-bugzilla-for-java/

Social Media Repositories

We have three again well-known social media channels in our study. All of them have their limitations such as IRC logs are hard to parse since user can essentially write in any format since there is no real specification for IRC. On the other hand, Twitter's REST API is working like a charm and it makes ingesting Twitter data much easier than the other social media repositories. Mailing archives ingestion is also a hard one since we have many mailing archive repositories and they differ from each other such as one is gzipped and the other one is not or one has body as text and other has body as Gzipped as well.

IRC

We have only ingested IRC logs for the case study of hibernate. Other case studies didn't have the IRC logs available. For initial step, we should be getting all the file names the repository. We have done it using Jsoup which is a Java html parsing library[16]. After getting all the files in a sorted manner then, next step was to parse a line of IRC log which is essentially a social media entry. Jsoup also came for the rescue and we have parsed the IRC line. Parsing IRC line was quite hard because there are be many cases such as joining, nick changing, and server messages. In addition, since there is no predefined language or protocol for IRC logs, it was quite hard to track the patterns for extracting nick names and/or even the actual message.

Mailing Archives

All of our case studies have mailing archives. One of them has storing files as gzipped without providing any API or anything useful. Thus, we have made web crawling once more using Jsoup and getting all files with that method. All these files have to be in sorted order because insertion to the database of social media repository entries matter for me. We should be keeping the order all the time in order to know which entry is a response to the other one or vice versa. After that step, we have a gzipped file to be parsed thus, we have Apache James's Mime4j which is quite useful Java library for parsing email's into Java objects[17]. On the other hand, same flow applies to the other case studies which provides mbox mail archive files. At this point only task left is to save the parsed email into database.

Twitter

We made searches for all the case studies on Twitter. Ingestion Twitter was the easiest among other ingestions due to Twitter's Rest API[18]. We have used a Java library called Twitter4j which is a wrapper library around Twitter's Rest API[19]. Only limitation here was being limited into 100 tweets per search and some other limitation which comes from Twitter's API rates. The rest is just to make the information that you get from twitter4j for saving into database.

[16] http://jsoup.org/
[17] http://james.apache.org/mime4j/index.html
[18] https://dev.twitter.com/rest/public
[19] http://twitter4j.org/en/index.html

4 Unified Models

For extracting data from both social media and issue tracking systems, given we have different projects, issue tracking tools and social media channels, we had to unify them into one then we can extract the data according to that model hence, we can analyse it much easier as well. For creating unified models for each sub-system, we used reverse engineering. We analysed each systems and unified the systems into models. In the unified models, there are some system specific attributes such as Twitter which provides us location information which we don't have in any other social media channel.

4.1 Issue Tracking Systems

We have two issue tracking systems that we need to extract data from. Although their models were almost identical to each other, we still had to make some work on unifying them into one model. This section describes how we combined these two issue tracking systems into one unified model.

JIRA

JIRA database schema is listed in the Annex II. JIRA database schema is essentially close to our unified model. They also have issues table as their main table and they also create the other table around the issues table. Most of our tables are common with them including custom fields. They also have projects, attachment, priority, issuelink, table like we do.

Bugzilla

Bugzilla database schema can be found in Annex III. Same as JIRA schema, Bugzilla schema is also somewhat close to us but not with same degree as JIRA. They also have bugs (issues for us and JIRA) table as pivot and created rest of the tables around bugs table. We only have attachments table in common regarding database schema but we do have all the useful information of them in our unified model but with a different schema architecture.

Unified Model for Issue Tracking Systems

As shown in Figure 4, we have unified Bugzilla and JIRA issue tracking systems into one model. We created Issues table is the main table in issue tracking systems unified model. For capturing user information, we created two more table called User and IssueRepositoryUser. IssueRepositoryUser table captures all the user information for a given issue and User table is present for both of the unified models. User table essentially links these two unified models with user information captured from social media channels and issue tracking systems. Since a user can comment in a given issue, we created Comments table for capturing comments. History table created for capturing history of a given issue such as updated assignee or changed priority. An issue can have issue links which means links to other issues such as issue x is required by issue y and for capturing this information, we created IssueLinks table. We created attachments table for capturing information about attachments but due to limitations, we are not able to fill this table with attachment information. In addition, issues can have project or company specific custom fields, thus, we have two tables for custom fields called CustomFieldValue and CustomField. We captured custom value field information in CustomFieldValue table and stored what actually the value is for custom field is CustomField table. Project and IssueRepository table is created for identifying which project or issue tracking system we are getting the data from.

Priorities table is created for capturing priority information for a given issue such as severe, minor, and major.

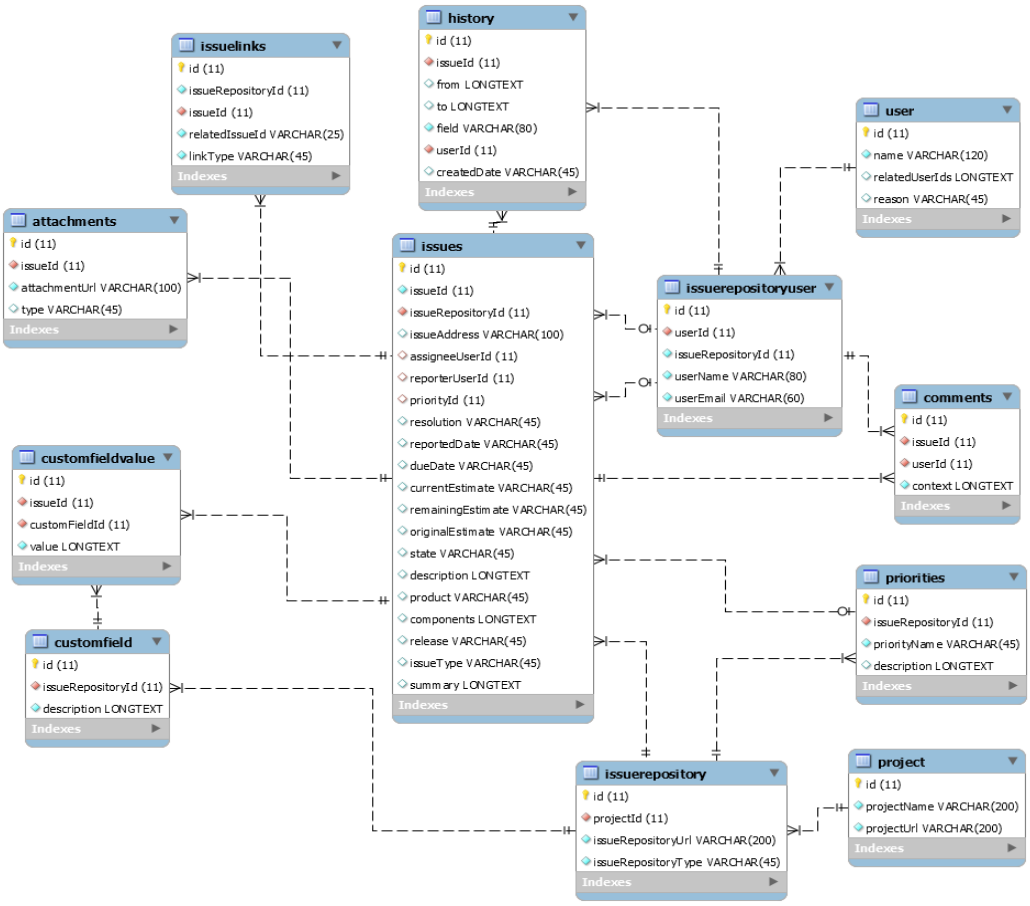

Figure 4. Issue Tracking System Unified Model

4.2 Social Media Channels

We have three social media channels IRC, Twitter, and email lists archives. Unlike issue tracking systems, social media channels are that similar to each other. This section describes how we combined these three social media channels into one unified model.

IRC

As shown in Figure 5, IRC reverse engineered model can be seen. This model contains content which is the message body, IRCcommand which is essentially shows the event type such as Join, Quit, and changing Topic or Nickname. Sender represent the sender of the given message and SentDate captures the sent date of a given message.

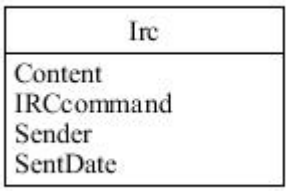

Figure 5. Reverse engineered IRC model.

Twitter

As shown in Figure 6, we have content again as message body, SentDate, and sender. In addition, we have Location for capturing where tweet had been sent from. InResponseTo field is for capturing if someone had commented given tweet and Hashtags are also for capturing what is the tweet about and subject field which identifies the subject of a given tweet.

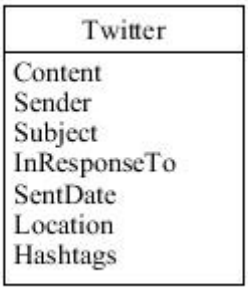

Figure 6. Reverse engineered Twitter model.

Mail Archives

As shown in Figure 7, we have similar things with Twitter and IRC such as Content and SentDate. In addition to them, we have AddressIds for capturing who sent, received, BCC and CC information. Subject field captures the subject of a given email. We also have couple of Booleans such as IsReplied, IsImportant, and IsSpam. IsSpam is for identifying if the email is spam or not, IsImportant captures if the email is important of not and IsReplied captures if the email is being replied or not. In this case we might have attachments as well and they are captured as AttachmentsIds. //TODO this text has to reflect the change if Siim is okay with the new model.

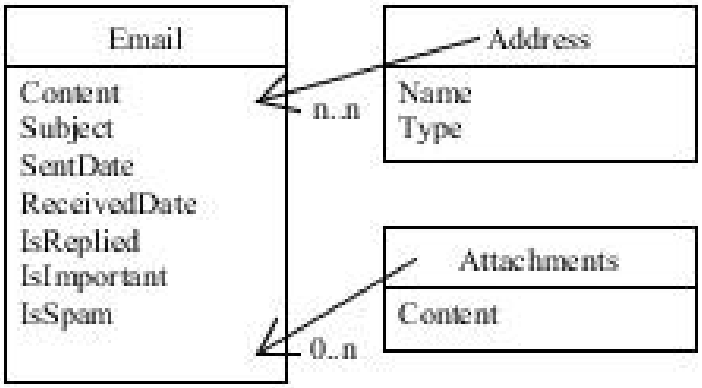

Figure 7. Reverse engineered email model.

Unified Model for Social Media Channels

As shown in Figure 8, SocialMediaEntries is the main table which we capture the general information about a given social media entry. In addition, SocialMediaRepository table is created for specifying which social media channel we are getting the social media entry

from. SocialMediaEvents table is created for capturing information such as IRC exit, IRC join, and IRC nick change information. Also, in order to capture user information same as the other unified model, we created two tables called User and SocialMediaUser. User table is containing a field called RelatedUserIds which we populate if we encounter a user nickname change or when the users are related.

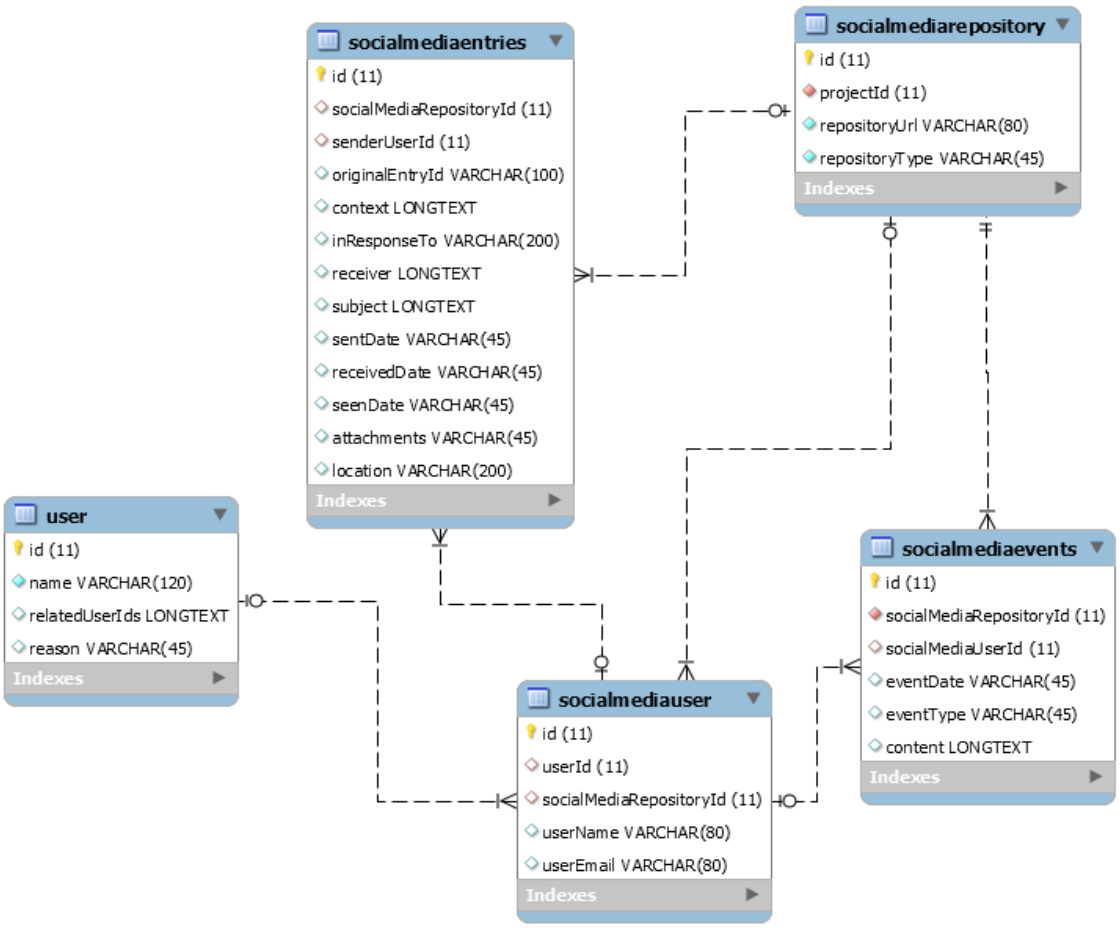

Figure 8. Social Media System Unified Model.

4.3 User Matching Lifecycle

There are two ways of creating a user in our system. If the user is not being encountered before then one of the flows will be happening depending on if an entry is social media entry or issue tracker entry:

Flow for social media entries:

Figure 9 shows that for ingesting a social media entry in our system, we should be creating a user for a given social media entry. When we have the social media entry data and then we can save user of a given entry to both SocialMediaUser table and User table.

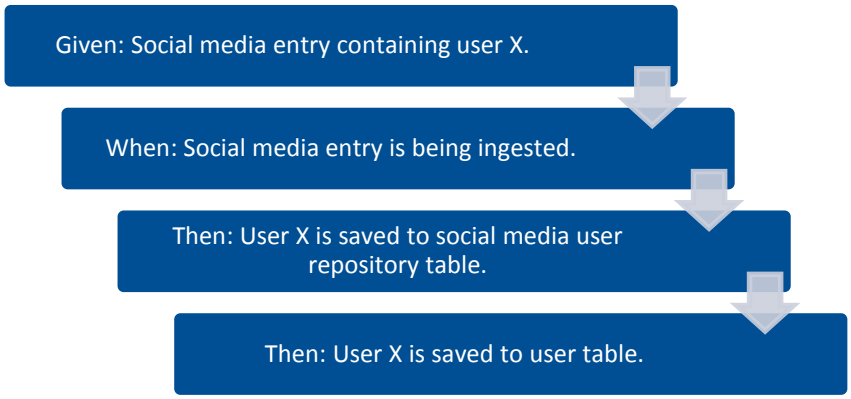

Figure 9. Social media entries flow

Flow for issue repository entries:

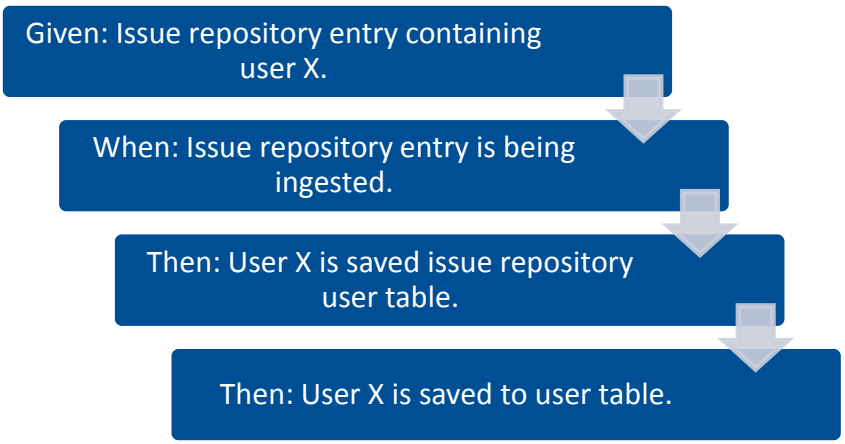

Figure 10. Issue repository entries flow

It can be seen from Figure 10 that in order to ingest an issue repository entry, we should be creating user of a given entry into tables IssueRepositoryUser and User.

5 Implementation

In this section, we are talking about implementation of adaptors which ingested both social media channels and issue tracking systems. In addition, overall architecture of our implementation as well as description about the implementation of adaptors.

5.1 Architecture Overview

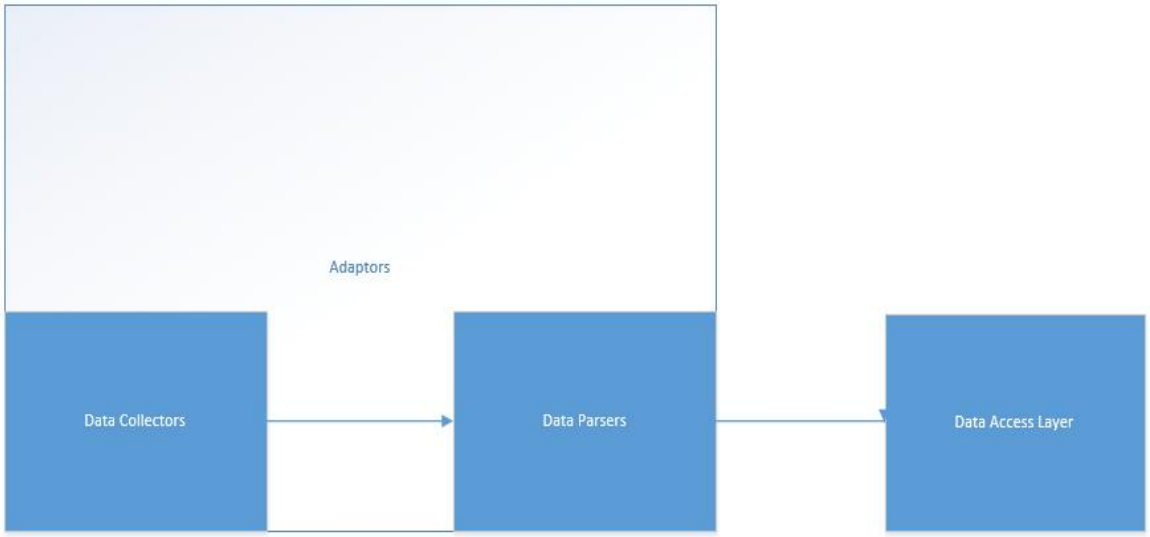

Figure 11. System Architecture Overview.

As shown in Figure 11, our system's architecture consist of three parts: Data Collectors, Data Parsers, and Data Access Layer. Data Collectors are collecting data from the repositories and dumping them into Java memory, then parsing are starting to do their role which is parsing these raw data into Java objects which are defined using unified models. For the last step, we are saving these parsed data into our database server using our Data Access Layer which essentially wraps MySQL.

5.2 Adapters

We created adaptors using Java and its libraries. These adaptors' implementation can be accessed from https://github.com/ilgun/unifiedissuetrackers. This repository is public and one can download and try them without any authentication. There are two main adaptors for issue tracking systems, one for JIRA and other one for Bugzilla. On the other hand, there are adaptors for social media such as Twitter, email archives, and IRC logs. There are two main unified model main methods for running the adaptors easily. One is called SocialMediaUnifiedMain.java which runs all available social media adaptors and the other one is called IssueRepositoryUnifiedMain which is for all available issue repository adaptors.

5.3 Data Access Layer

Data that we collected via adapters are stored in MySQL[20] database which is using InnoDB[21] as a storage engine. We haven't had much options when it comes to database engine selection and InnoDB is considered a wise choice if the concern is performance and wide selection of features[21]. JDBC is used for connecting to our MYSQL database from

[20] http://dev.mysql.com/
[21] http://dev.mysql.com/doc/refman/5.5/en/innodb-introduction.html

Java code. JDBC stand for Java Database Connectivity from Oracle Corporation. Essentially, JDBC works as database independent which means you should be able to connect to any database server with JDBC[22].

5.4 Implementation Details & Problems Faced

In this section, there are some several subsections for explaining the faced difficulties while implementing the adapters.

JIRA Getting All Issue Ids

JIRA REST API is useful in many aspects however, to get list of issues, it is limited to 100. It provides total number of issues can be given a query so that it is possible to loop with a starting value in the query e.g. `'startAt=0'` till it reaches the total issue count.

JIRA Changelog Json Expansion

We are getting the history of a given issue by expanding the original issue JSON with a value of `'expand=changelog'`. However, for versioning reason some of the projects didn't allow us to get the changelog from original JSON. Thus, we had to get original issue id from JSON and make one more call to the server for getting changelog.

Bugzilla Getting All Issue Ids

Bugzilla REST API provides this function however, some of our projects' bugzilla versions didn't support REST API. On that case we have to inject all issue ids manually from a file.

Bugzilla REST API issues

Since some projects didn't support REST API, then we had to rewrite most of the function for extracting data. Then we have used RPC API but it was not fully returning what we should have extracted. Then, some web crawling work had to be made.

Mail Archive Parser

Some of the projects' email archives supported mbox format, then is it easier because you get all the email archives. However, problematic part starts with gzipped email archive file. It is problematic because some of them are not really gzipped and some of them are. Thus, you need to make your implementation to support both of them. In this case it was quite okay because to understand if a file is gzipped or not is quite easy in Java. This check can be done by simply checking the header of a file.

One other issue was to extract these files from web. Then, web crawling done for getting the file path. For both email archives retry methods had to be implemented since the connection to the server were not reliable and thus, they were throwing exceptions for no reason.

Actual email parsing was also problematic because there are many projects that we extracted the email archives. Although, we have used a nicely implemented Java email parsing library sometimes it was not successful for some projects, hence, we had to write our own parsing implementation for these cases.

[22] http://www.oracle.com/technetwork/java/javase/jdbc/index.html

IRC Log Parser

Parsing IRC logs was quite tricky because there were many different versions of files. Files were not consistent because some of them were already parsed by some kind of application. For getting these files, web crawling techniques used and for parsing them we have implemented an application which is written in Java.

Nickname Change

While parsing IRC logs, we have faced a big problem. The problem was to identify which user is the main user when a nickname change event is encountered. With a naïve approach, we could store both users in our database, however, this approach would backfire when we are trying to match social media users with issue repository users. Thus, we were in need of an approach which would give us the knowledge of the main user. Our solution represented in below Figure 12.

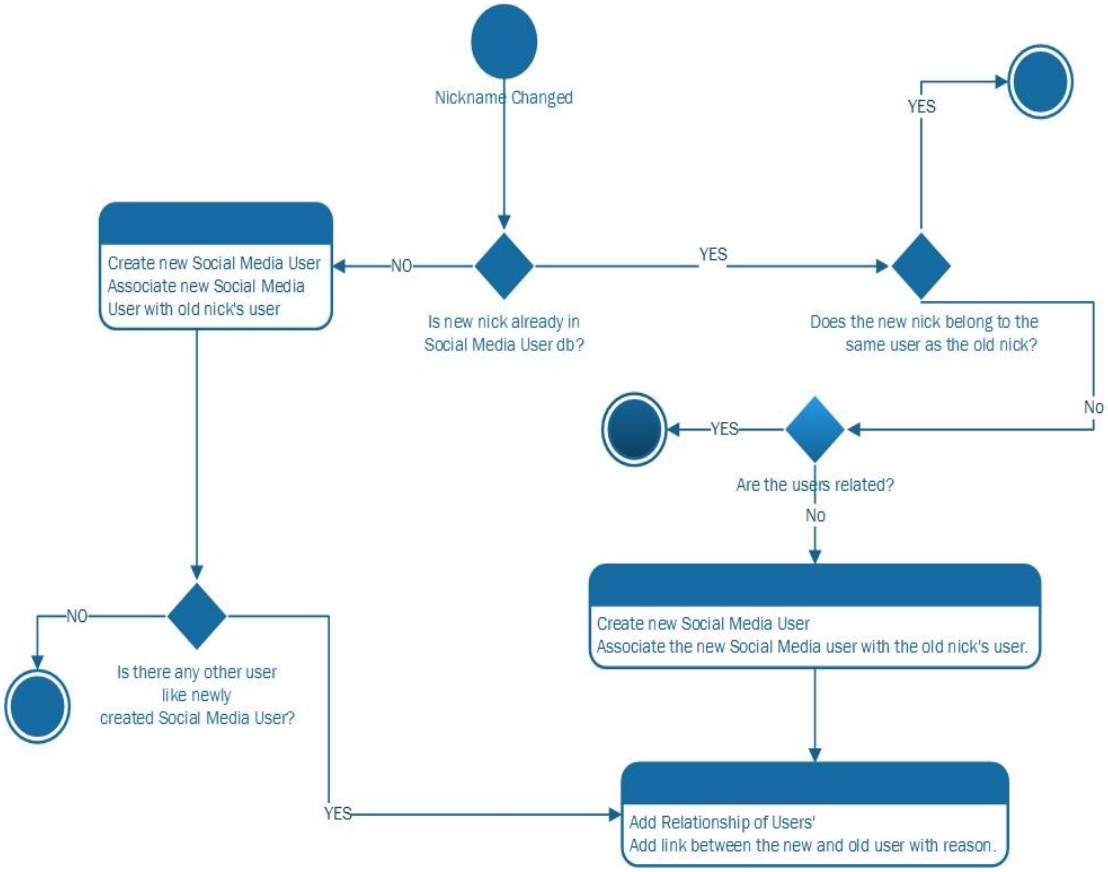

Figure 12. Nickname change user flow.

Nickname Change Use Cases

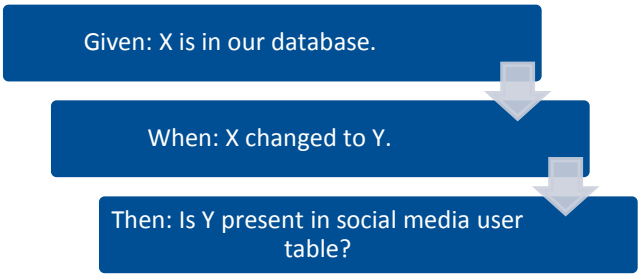

Figure 13. Nickname change flow

After this point Figure 13, we might have two different flows:

Possible Flow 1: Y is not present in social media user table

In this case we will create a new social media user for Y. Then, associate the new social media user with userId of old user. If there are any other users like that in our database, we will find them all and add relation with user.

Possible Flow 2: Y is present in social media user table

If Y does not belongs to the X's user and users do not have relationship in our database already, then we will create new social media user for Y. Also, associate it with X' user. Finally, we will add a relationship between X's and Y's user.

6 Analysis Examples

In this section, we are presenting our analysis that we made based on the data that we collected from our adaptors. These questions are demonstrating what we can gain with combining social media channels with issue tracking systems.

A) If users in IRC change nicks in Hibernate, are they actively using JIRA?

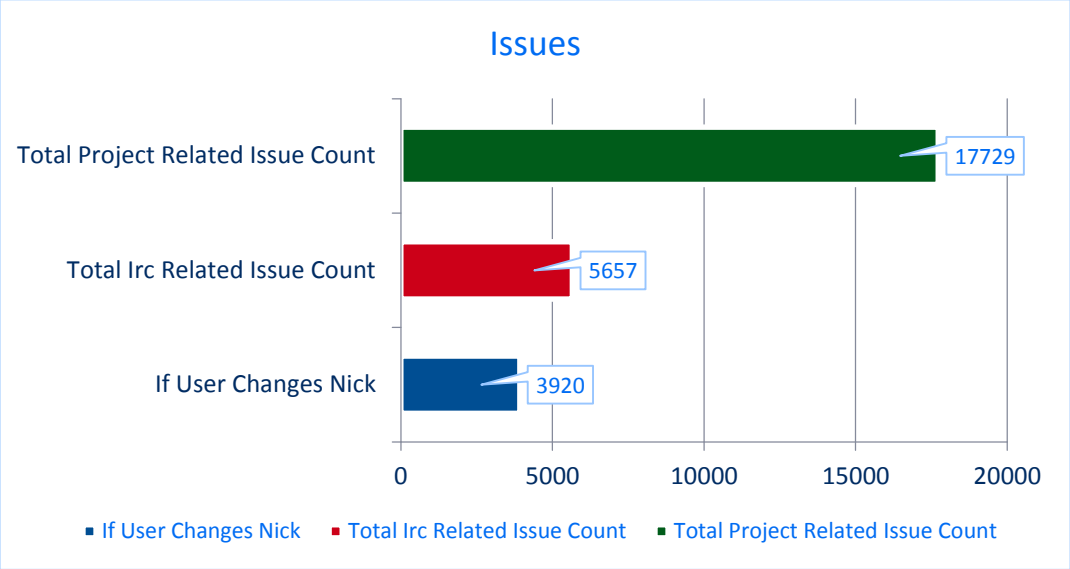

Figure 14. If IRC user change nicks in Hibernate, are they actively using JIRA

We just can take Hibernate project into account because IRC is only used by Hibernate developers. Total IRC related issue count shows the number of users who wanted their nicks in IRC and also was active in JIRA. As shown in Figure 14, only 31% of the total number of issue records in our database are related with IRC social media channel. Out of this 31%, around 69% are the users who are changed their nick at some point in IRC. Total project related issue count is 17729 and 5657 of them are related with IRC. We can interpret that if a user changes nick in IRC, it is likely to be a user who is active also in issue tracking systems. Query used for this analyses can be found from SQL Query 1.

B) Which social media channel is more actively used in Hibernate?

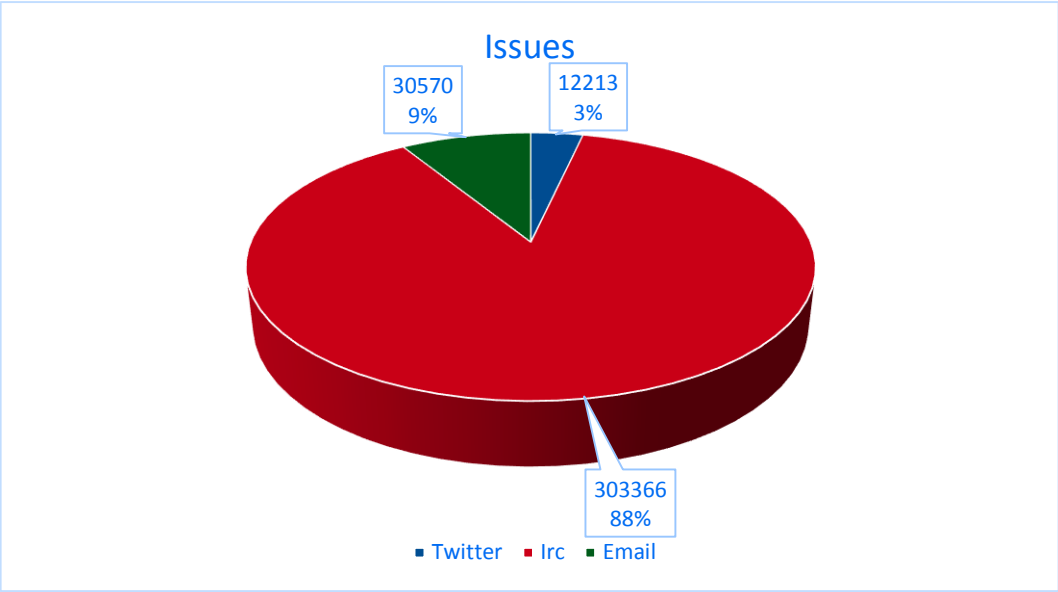

Figure 15. Which social media channel is more actively used

According to our Hibernate case study, users are much more active in IRC 88% than other social media channels such as Twitter 4% or email 9%. Actually, although the percentage is so highly in favour of IRC, we should be keeping that in mind, that users who communicate through IRC are tend to send multiple entries easily. Such as they might say hi and after a question or statement. However, for Twitter or more likely in email communication, users are likely to send one email per one topic. On the other hand in IRC, users are tend to write one more than one entry for a topic (this can incredibly high). Query used for this analyses can be found from SQL Query 2.

C) If a user is active in IRC, then is he also active in JIRA?

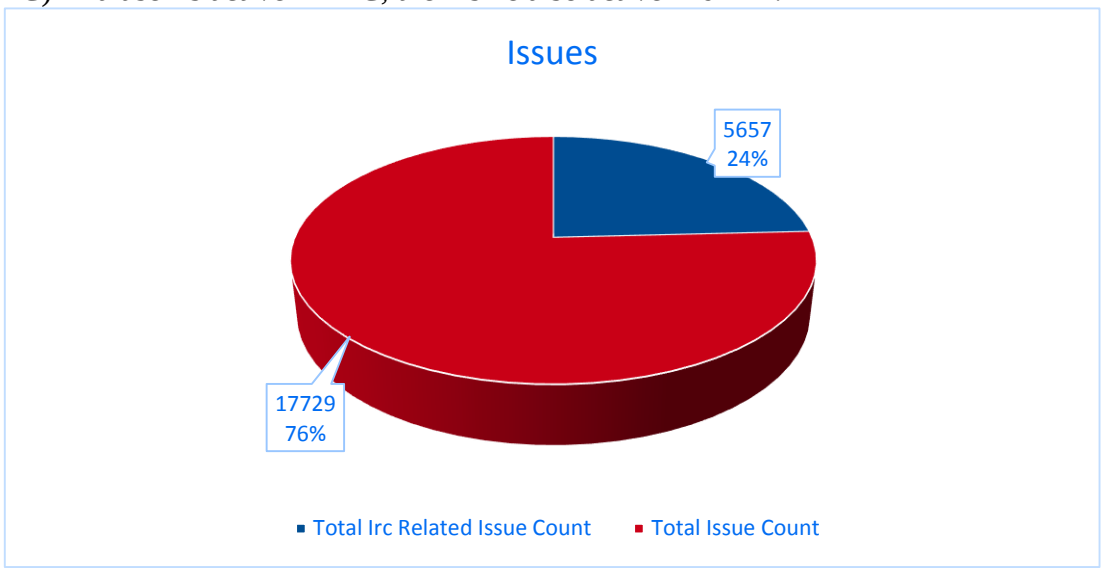

Figure 16. If user is active in IRC, is she/he also active in JIRA

Only Hibernate project is used for this query because IRC is only used in Hibernate in our case studies. As shown in Figure 16, IRC related issue entries total number is 24% percent of the total issue entries which comes from the project with IRC social media channel used. This number can be seem relatively small however, it actually means 24% of the issue entries are discussed in IRC social media channel which is greater number. Query used for this analyses can be found in SQL Query 3.

D) If a user changes nick in IRC, then is he reporter or assigner in JIRA?

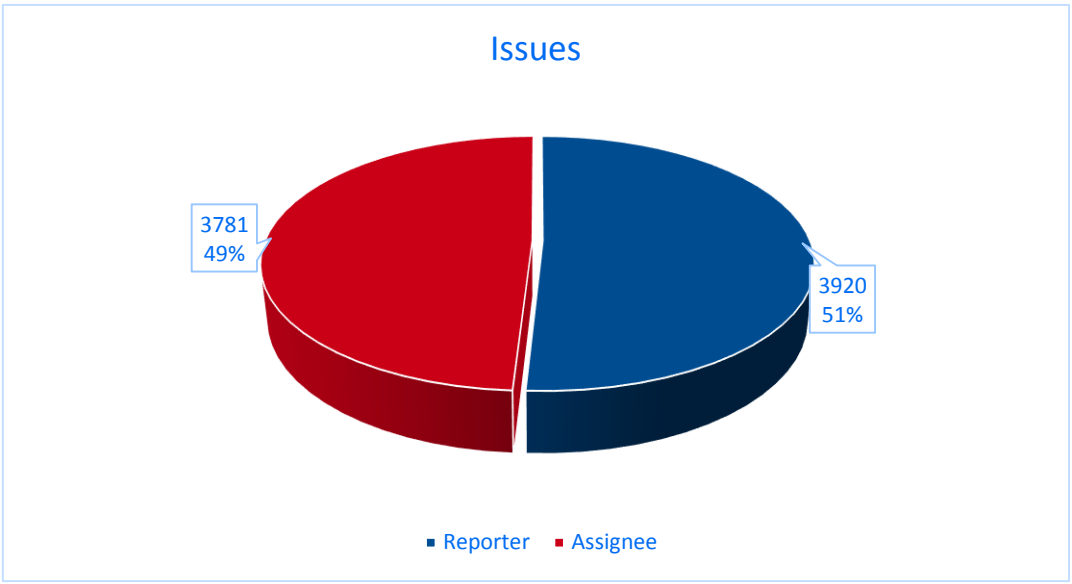

Figure 17. If user changes nick, is she/he more likely to be reporter or assignee in issue tracking systems

As shown in Figure 17, if a user changes nick in IRC, then 51% that she/he is a reporter and 49% that she/he is to be an assignee. We can take into account only Hibernate project in this case as well simply because we don't have IRC social media channels for other case studies rather than Hibernate. From this graph, we can interpret that users are in IRC social media channel can be both assignees and reporters. Query used for this analyses can be found from SQL Query 4.

E) If a user replies to email archives, is he reporter or assigner?

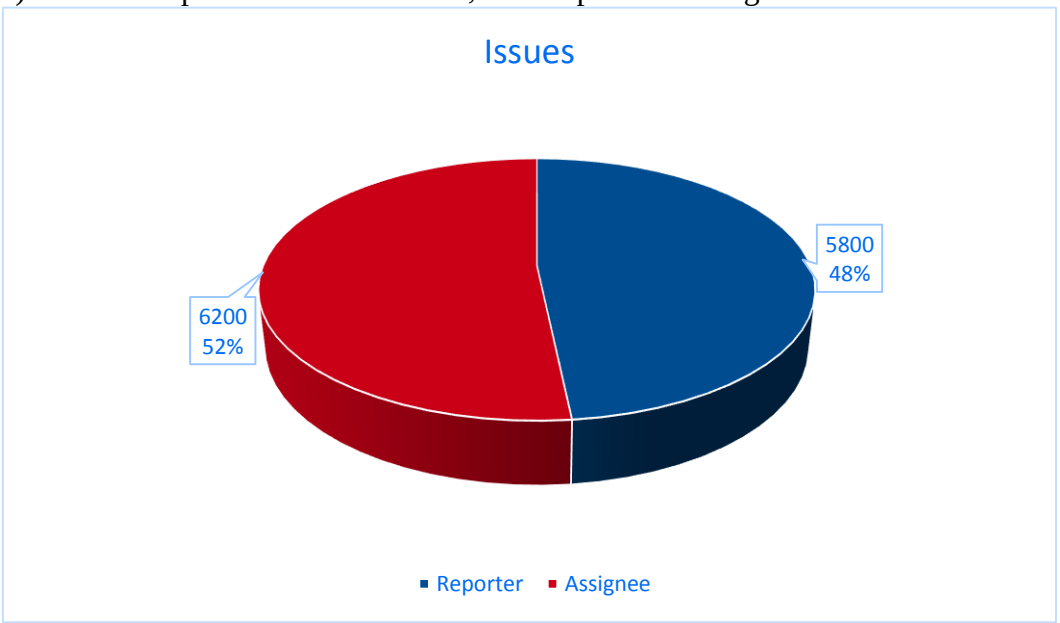

Figure 18. If user replies to email archive threads, is she/he more likely to be reporter or assignee

As shown in Figure 18, if a user replies to email archives, then 52% that she/he is an assignee in an issue, 48% that she/he is to be a reporter. Same as IRC, we can interpret that users are in email social media channel can be both assignees and reporters. Unlike IRC, all of our case studies contain email archives so that, this graph represents the total number

of all emails that we found in the case studies. Query used for this analyses can be found from SQL Query 5.

F) If a user is active in email archives then, is he active also in issue tracking systems?

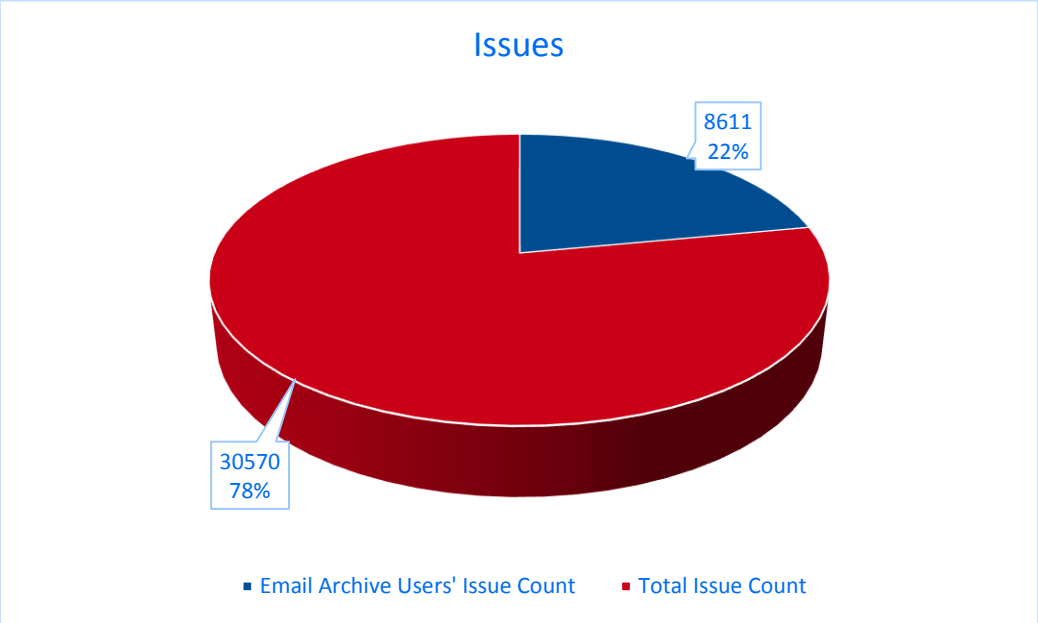

Figure 19. If user is active in email archive threads, is she/he also active in issue tracking systems

As shown in Figure 19, email related issue entries total number is 22% percent of the total issue entries. Same as IRC, this number can be seem relatively small however, it actually means 22% of the issue entries are discussed in email communication of all case studies since we have email archives for all case studies. Query used for this analyses can be found from SQL Query 6.

G) If a user is active in Twitter then, is he active also in issue tracking systems?

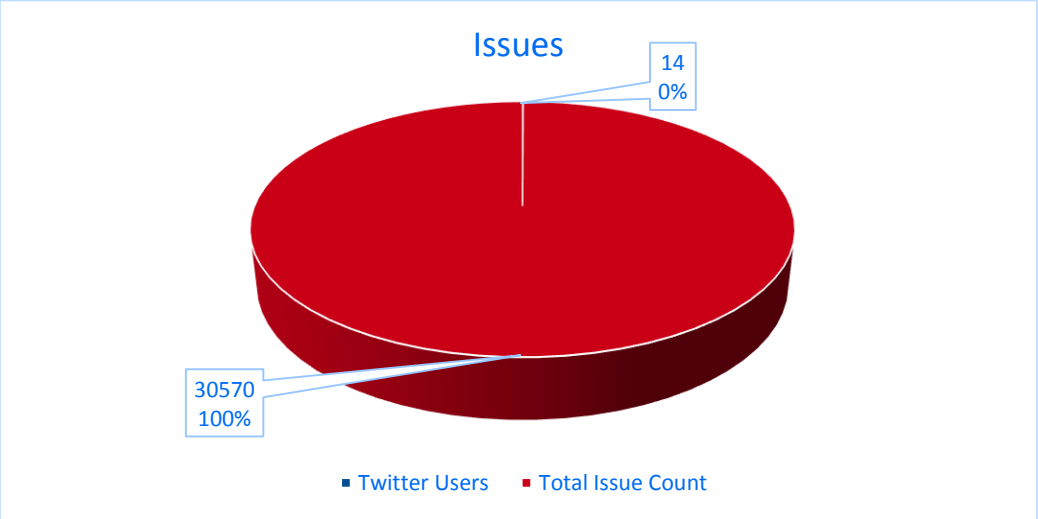

Figure 20. If user is active in Twitter, is she/he also active in issue repository

As shown in Figure 20, there is not even 1% relationship between Twitter and issue tracking systems. Users of Twitter do not tend to be in software development cycle. Generally Twitter is used for announcements for software and users are generally preferring IRC or Email for their communication channels. We searched for Tweets for all case studies ex-

cept Pulp because nothing useful could be obtained from that. Query used for this analyses can be found from SQL Query 7.

H) If user joins more than 10 times in IRC, if he active in JIRA?

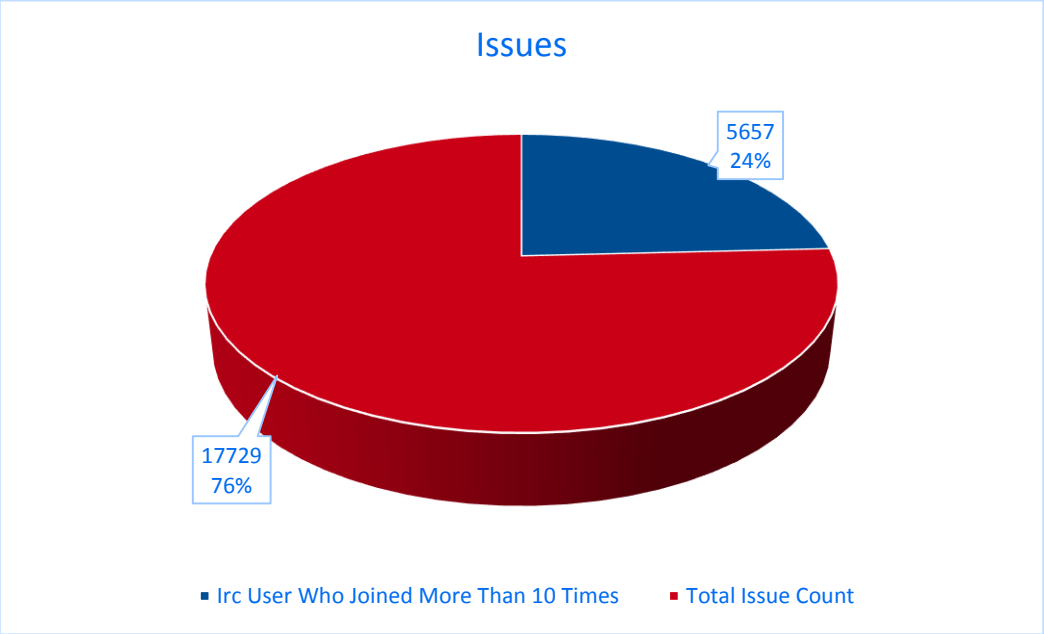

Figure 21. If user joins more than 10 times in IRC, is she/he also active in issue tracking systems

As shown in Figure 21, IRC users are almost 100% changes their nicks and 24% they are active in issue tracking systems. This query is valid only for Hibernate since we don't have the other case studies supporting IRC social media channel. If an IRC social media channel user is joined to IRC more than 10 times than he is likely to be active also in JIRA. In addition, shown in Figure 16 issue count found for a user who joined IRC more than 10 times and issue count for every IRC user who had used JIRA are identical(5657). We can interpret this as IRC users who are also present in JIRA are 100% join IRC more than 10 times. Query used for this analyses can be found from SQL Query 8.

I) If user joins more than 10 times in IRC, is he more likely to be a reporter or assignee in JIRA?

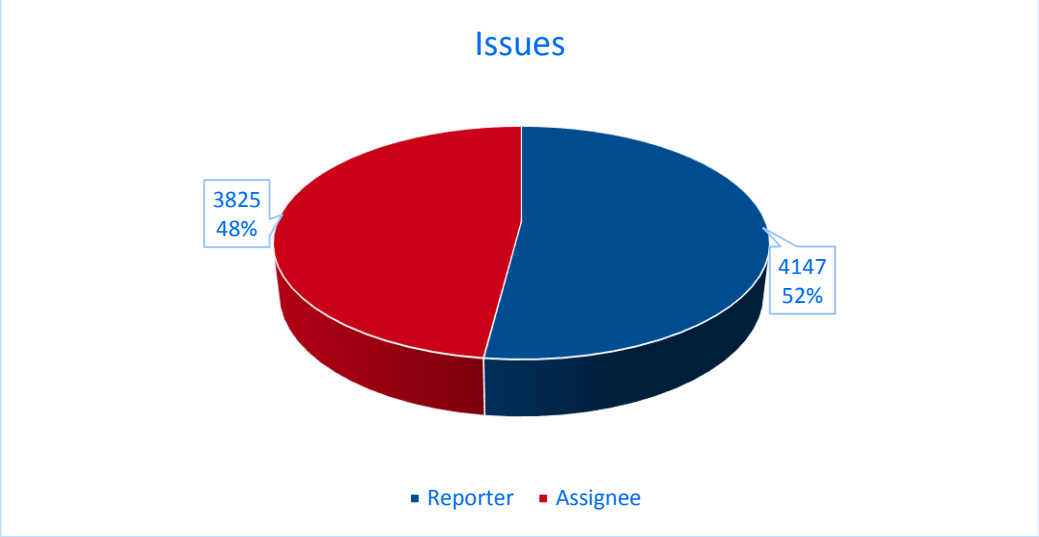

Figure 22. If user joins more than 10 times in IRC, is she/he more likely to be a reporter or assignee

As shown in Figure 22, if an IRC users changes nick then, he is 52% likely to be a reporter and 48% likely to be an assignee in issue tracking systems. Query used for this analyses can be found from SQL Query 9 and only valid for Hibernate case study.

J) If user quits more than 20 times in IRC, is he active in JIRA?

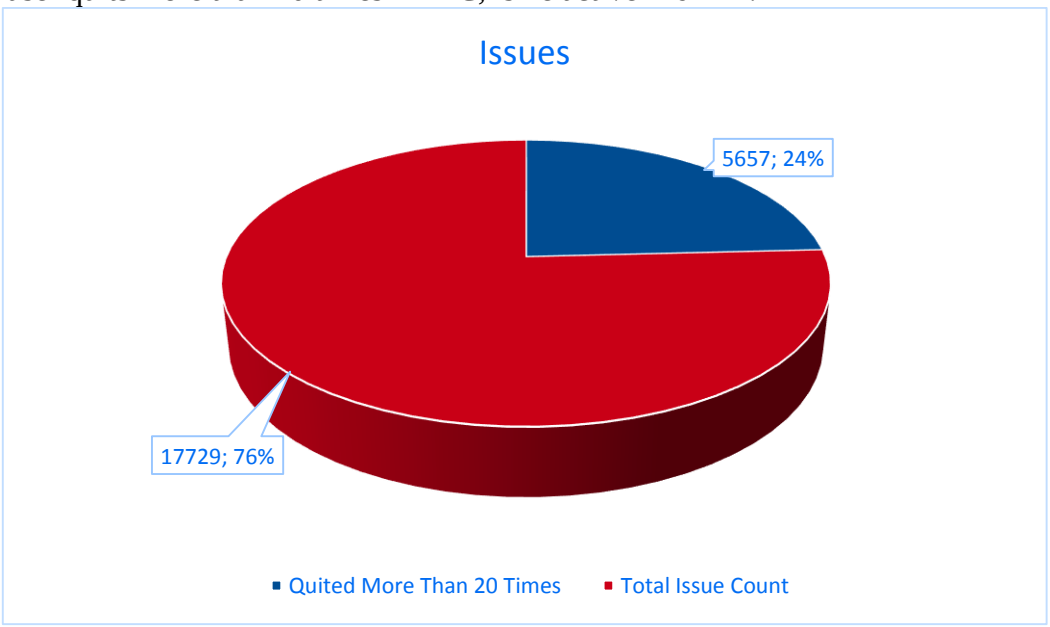

Figure 23. If user quitted more than 20 times in IRC, is she/he is also active in issue tracking systems

As shown in Figure 23, it is essentially the same percentage (24%) as total IRC related issue entries in issue tracking systems, so that we can interpret that, all IRC users are quitted IRC more than 20 times. Shown in Figure 16 and Figure 21, IRC users who quitted more than 20 times, who joined more than 10 times and total number of IRC users who are present also in JIRA are equal. So that, we can interpret this as IRC users in Hibernate case study who had used JIRA are 100% quitted IRC more than 20 times and also joined IRC more than 10 times. Query used for this analyses can be found from SQL Query 10.

K) If user quits more than 20 times in IRC, is he more likely to be reporter or assignee in JIRA?

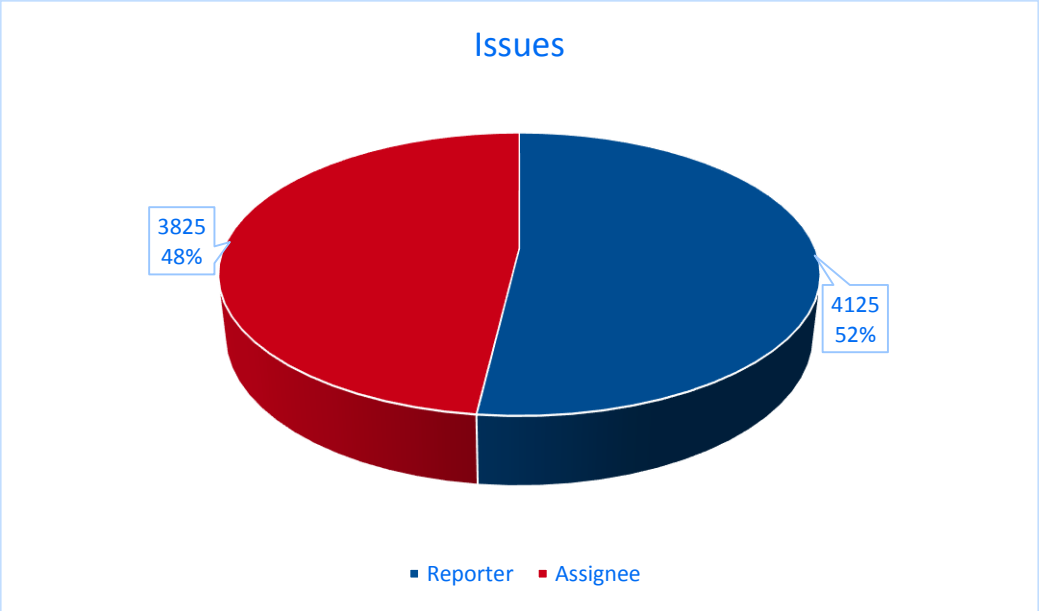

Figure 24. If user quits more than 20 times in IRC, is she/he more likely to be reporter or assignee

As shown in Figure 24, there is 52% chance for a given user to be a reporter in issue tracking systems, with 48% chance of being assignee. Since the ratio is pretty similar, we can state that both assignees and reporters are using IRC actively for their daily developer discussion. Query used for this analyses can be found from SQL Query 11 and query is valid only for Hibernate case study.

7 Related Work

There are many studies which involves mailing list mining and analysis such as [11, 12] which tried to get an answer to the question of "how social networks evolve in open source projects world".

A prescript of analysing newbies' behaviours in open source community and their first interactions, they were able to show that newbies are receiving swift replies to their first posts and flaming is common in open source software world especially for woman and minorities [13].

Rigby and Hassah analysed psychometric text on apache developer mailing list, their main goal was to assess the usefulness of the linguistic inquiry and Word Count tool as a predictor and understanding open source software development. They also analysed personality and releases of the apache foundation and its effect on personalities as well as analysing new, current, and departing developers with LIWC dimensions for their first and last emails [14].

Shihab et al studied IRC social media channel called `#gtk-devel`, based on IRC meetings and they found out that the length of IRC meetings is increasing over time, IRC meetings have a positive effect attendance and its increasing also over time, IRC meeting contributors are actively contributing to IRC meetings [15].

Bettenburg et al identified challenges when it comes to ingest mailing list archives. They showed that there are no automated perfect solution for ingesting mailing list archives if the email messages are different than each other [16].

Ayari et al studied Bugzilla versioning control system and issue tracking systems integration problems, they merged data from Bugzilla CVS repositories and issue tracking systems based on IssueIds [17].

In addition, there are studies about issue tracking systems such as [18] which tries to investigate how communication of the teams effect the software development process and he found out five team roles such as issue tracker as a repository for organizational knowledge, the issue tracker and its boundary objects that bounds the stakeholders together, the issue tracker as a communication hub for different sides of software development process, the issue tracker as persistent, asynchronous and oftentimes multicast communication channel, and the issue tracker as a contextualization repository providing that a canonical address for all the related things together of issues. He also listed identifications of seven considerations of issue tracking systems. Bertram et al analysed the social nature of issue tracking systems in small, collocated teams [19].

Moreover, paper by Bosu proposes a research plan for mining open source repositories for revealing community structures of it [1]. A recent study which aims to find out communication perspective of an open source software project, found out that email threads cover a wide range of topics and implementation details are only in a portion of them, that code artifacts are also mentioned in topics not related to implementation, and that project developers are not the majority of the participants and also they have found some evidence about mailing lists are not only for developers discussing implementation details [20].

However, integration between social media and issue tracking systems were missing, thus we are attempting to integrate these two and analyse the results of it.

8 Conclusions

This paper is set out to explore what we can get when we combine social media channels with issue tracking systems. Three case studies selected for the study, these case studies are selected from known open source software projects such as Hibernate and Hive. In order to get data from issue tracking systems, we have created unified models so that we can get the data and analyse it effectively. In addition, a unified model is also created for social media channels for the same purposes. We have shown that the combined unified model can answer questions such as:

1) If a user is active in email archives, is he active also in issue tracking systems?
2) Which social media channel is used more actively?
3) If a user replies to email archives, is he likely to be a reporter or an assigner?
4) If an IRC user changes nickname, are they also active in JIRA?
5) If a user changes nick in IRC, is he reporter or assigner in JIRA?
6) If users are active in Twitter, are they also active in issue tracking systems?

Some of our main contributions are:

1) Unified model created for social media channels.
2) Unified model created for issue tracking systems.
3) Adapters created in order to get data for analysis.

We were successful in combining the data for all social media channels and issue tracking systems. We ingested data from well-known social media channels such as Twitter and IRC and also well-known issue tracking systems Bugzilla and JIRA. Adaptors implemented in Java for getting, parsing, and storing data.

Some results that we found are:

1) If a user is active in email archives, he is likely to be active also issue tracking systems.
2) IRC is much more used compared to other social media channels such as Twitter and email archives
3) Our study states that, the ratio is quite similar which 52% assignee, 48% reporter is.
4) If a user changes nickname in IRC, he is likely to be active in JIRA.
5) If a user changes nickname is IRC, he has 51% chance to be a reporter and 49% chance to be an assignee in JIRA.
6) If a user is active in Twitter, he is not likely to be part of any issue tracking systems.

Analysing social media channels combined with issue tracking system had never done before with our perspective thus, we opened up a new insight on OSS.

We focused on user perspective in our study, one can focus on iterations and products of a given projects which would lead to analyse software methodologies and its effects on software development.

9 References

[1] A. Bosu, „Mining repositories to reveal the community structures of Open Source Software projects,“ Tuscaloosa, AL, USA.

[2] D. C. a. K. S. Booth, „Coordinating open-source software,“ in *Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1999.

[3] A. Khanjani und R. Sulaiman, „The Process of Quality Assurance under Open Source Software Development,“ in *IEEE Symposium on Computers & Informatics*, 2011.

[4] D. M. German, D. Cubranić und M. A. D. Storey, „A framework for describing and understanding mining tools in software development,“ in *MSR '05 Proceedings of the 2005 international workshop on Mining software repositories*, ACM New York, NY, USA, 2005.

[5] „Twitter Information,“ Twitter Inc, February 2015. [Online]. Available: https://about.twitter.com/company. [Zugriff am 03 February 2015].

[6] Reis, C.R. and de Mattos Fortes, R.P, „An overview of the oftware engineering process and tools in the Mozilla project.,“ *Workshop Open Source Software Development, U*, p. 155–175, 2002.

[7] C. Henderson, Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications, Sebastool, CA: O'Reilly Media, 2006.

[8] R. Black, Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, New York: Wiley Publishing, 2002.

[9] „Bugzilla issue lifecycle,“ Bugzilla, 3 February 2015. [Online]. Available: https://bugzilla.readthedocs.org/en/latest/using/editing.html#life-cycle-of-a-bug. [Zugriff am 3 February 2015].

[10] „Jira Workflow,“ Atlassian, 3 February 2015. [Online]. Available: https://confluence.atlassian.com/display/JIRA/What+is+Workflow. [Zugriff am 3 February 2015].

[11] L. Yu, S. Ramaswamy und C. Zhang, „Mining Email Archives and Simulating the Dynamics of Open-Source Project Developer Networks,“ in *Proceedings of EOMAS*, 2008.

[12] C. Bird und A. Swaminathan, „Mining Email Social Networks,“ MSR, Shanghai, China, 2006.

[13] C. Jensen, S. King und V. Kuechler, „Joining Free/Open Source Software Communities: An Analysis of Newbies' First Interactions on Project Mailing Lists,“ in *Proceedings of the 44th Hawaii International Conference on System Sciences*, Hawaii, 2011.

[14] P. C. Rigby und A. E. Hassah, „What can OSS mailing lists tell us? A preliminary psychometric text analysis of the Apache developer mailing list,“ in *Fourth International Workshop on Mining Software Repositories*, 2007.

[15] E. Shihab, Z. M. Jiang und A. E. Hassan, „On the use of IRC channels by developers of the GNOME GTK+ open source project,“ Kingston, ON, K7L 3N6, Canada, 2009.

[16] N. Bettenburg, E. Shihab und A. E. Hassan, „An Empirical Study on the Risks of Using Off-the-Shelf Techniques for Processing Mailing List Data,“ Kingston, Canada, 2009.

[17] K. Ayari, „Threats on Building Models from CVS and Bugzilla Repositories: the Mozilla Case Study“.

[18] D. Bertram, „The Social Nature of Issue Tracking in Software Engineering,“ Calgary, Alberta, Canada, 2009.

[19] D. Bertram, A. Voida, S. Greenberg und R. Walker, „Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams,“ Calgary, Alberta, Canada, 2010.

[20] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger und A. V. Deursen, „Communication in Open Source Software Development Mailing Lists,“ 2013.

Appendix

I. Queries used for Analysis

SQL Query 1

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM issuetrackers.socialmediaevents
WHERE eventType LIKE "NICKCHANGE"
))))
UNION DISTINCT
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM issuetrackers.socialmediaevents
WHERE eventType LIKE "NICHCHANGE"
)))))
```

SQL Query 2

```
SELECT count(id) FROM socialMediaEntries where socialMediaRepositoryId = 3
UNION
```

```
SELECT count(id) FROM socialMediaEntries where socialMedi-
aRepositoryId = 4
UNION
SELECT count(id) FROM issuetrackers.socialmediaentries where
socialMediaRepositoryId = 5 AND subject LIKE "%hibernate%"
```

SQL Query 3

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 4
)))
UNION DISTINCT
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 4
))))
```

SQL Query 4

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
```

```
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM issuetrackers.socialmediaevents
WHERE eventType LIKE "NICKCHANGE"
)))))
UNION
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM issuetrackers.socialmediaevents
WHERE eventType LIKE "NICKCHANGE"
)))))
```

SQL Query 5

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 1 or 2  or 3
))))
UNION
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
```

```
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 1 or 2  or 3
))))
```

SQL Query 6

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 1 or 2  or 3
)))
UNION DISTINCT
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 1 or 2  or 3
))))
```

SQL Query 7

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
```

```
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 5
)))
UNION DISTINCT
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT DISTINCT userId FROM socialmediauser
WHERE socialMediaRepositoryId = 5
))))
```

SQL Query 8

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM is-
suetrackers.socialmediaevents
WHERE eventType LIKE "JOIN"
HAVING userId > 10
))))
UNION DISTINCT
SELECT id FROM issues
```

```
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM is-
suetrackers.socialmediaevents
WHERE eventType LIKE "JOIN"
HAVING userId > 10
)))))
```

SQL Query 9

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM is-
suetrackers.socialmediaevents
WHERE eventType LIKE "JOIN"
HAVING userId > 10
)))))
UNION
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
```

```
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM issuetrackers.socialmediaevents
WHERE eventType LIKE "JOIN"
HAVING userId > 10
)))))
```

SQL Query 10

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM issuetrackers.socialmediaevents
WHERE eventType LIKE "PART"
HAVING userId > 20
))))
UNION DISTINCT
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM issuetrackers.socialmediaevents
WHERE eventType LIKE "PART"
```

```
HAVING userId > 20
)))))
```

SQL Query 11

```
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE reporterUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM is-
suetrackers.socialmediaevents
WHERE eventType LIKE "PART"
HAVING userId > 20
)))))
UNION
SELECT DISTINCT count(id) FROM issues
WHERE id IN (
SELECT id FROM issues
WHERE assigneeUserId IN(
SELECT id FROM issuerepositoryuser
WHERE userId IN(
SELECT id FROM user
WHERE id IN(
SELECT userId FROM socialmediauser
WHERE id IN (
 SELECT DISTINCT socialMediaUserId as UserId FROM is-
suetrackers.socialmediaevents
WHERE eventType LIKE "PART"
HAVING userId > 20
)))))
```

Database Models

Issue Tracking Systems

Issues Table

This is the main table which we store the issue entry.

Id	This field represents the unique id of a given issue entry. Our database assigns this id to a given issue with FIFO order. This field is common for both issue tracking systems.
IssueId	This field represents the original id of a given issue entry. This field is common for both of issue tracking systems.
IssueRepositoryId	This field represents the corresponding issue repository for a given issue entry. This field is common for both issue tracking systems.
IssueAddress	This field represents original issue address of a given issue entry. In other words this field represents the address of a given issue entry that we extracted data from. This field is common for both issue tracking systems.
AssigneeUserId	This field is common for both issue tracking systems, if a given issue is assigned to a particular user, then this field is filled with an id that represent the assignee. This id comes from our IssueRepositoryUser table. If a given issue is not assigned to anyone, then field is null in our database. This field is common for both issue tracking systems.
ReporterUserId	Almost same with the AssigneeUserId field, this field is also common for both issue tracking systems, this field represents the reporter of a given issue entry. However, this field cannot be null because every issue entry have a reporter. This field is filled with the id which comes from IssueRepositoryUser table. This field is common for both issue tracking systems.
PriorityId	This field represent the priority value of a given issue entry. We have priorities table for extracting this id because for different projects there can be different definitions. In other words 1 can means highest priority for a project but for another one, it can mean the lowest. Thus, we are saving descriptions for priority for a given project and returning the id from priorities table for this concern. This field is common for both issue tracking systems however, in Bugzilla it is called importance.
Resolution	This field represent the resolution for a given issue entry. This field is common for both issue tracking systems.
ReportedDate	This field represent the reported date for a given issue entry. Although this field is common for both issue tracking systems, in

	JIRA it is called created date.
DueDate	This field represent the due date for a given issue entry. Although this field is common for both issue tracking systems, in Bugzilla it is called deadline.
CurrentEstimate	This field represents the current estimation time for issue to be resolved. This field is common for both issue tracking systems.
RemainingEstimate	This field represents the remaining estimation time for issue to be resolved. This field is common for both issue tracking systems.
OriginalEstimate	This field represents the original estimation time for issue to be resolved. This field is common for both issue tracking systems.
State	This field represent the status of a given issue entry. In both issue tracking systems it is called status.
Description	This field represent the description of a given issue entry. This field is common for both issue tracking systems.
Product	This field represent the description of a given issue entry. This field is common for both issue tracking systems, however, it is called as project in JIRA.
Components	This field represent the components of a given issue entry. This field is common for both issue tracking systems.
Release	This field identifies the release version of the issue. This field is common for both issue tracking systems.
IssueType	This field represents the type of a given issue. This field is represented as type in JIRA and in bugzilla this field is not represented.
Summary	This field represents the summary of a given issue. This field is common for both issue tracking systems.

IssueLinks Table

This is the table which we store issue links.

Id	This field represents the unique id of a given issue link. Our database assigns this id to a given issue link with FIFO order.
IssueRepositoryId	This field represents the corresponding issue repository for a given issue entry.
IssueId	This field represents the related unique issue id.
RelatedIssueId	This field represents the related original issue id.

LinkType	This field represents the type of the link with the related issue.

Attachments Table

This is the table which we store attachments.

Id	This field represents the unique id of a given attachment. Our database assigns this id to a given attachment with FIFO order.
IssueRepositoryId	This field represents the corresponding issue repository for a given attachment.
AttachmentUrl	This field represents the original attachment's URL.
Type	This field represents the type for a given attachment.

CustomFieldValue Table

This is the table which we store the custom fields' values. Custom fields are optional because they are just present in JIRA.

Id	This field represents the unique id of a given custom field value. Our database assigns this id to a given custom field value with FIFO order.
IssueId	This field represents the related unique issue id.
CustomFieldId	This field is coming from CustomField table which identifies the custom field's description.
Value	This field represents the value of a custom field.

CustomField Table

This table is used to capture the custom fields' description.

Id	This field represents the unique id of a given custom field. Our database assigns this id to a given custom field with FIFO order.
IssueRepositoryId	This field represents the corresponding issue repository for a given custom field.
Description	This field represents the description of a given custom field.

History Table

This table is used to capture the activity of a given issue.

Id	This field represents the unique id of a given issue activity. Our database assigns this id to a given activity with FIFO order.
IssueId	This field represents the related unique issue id.
From	This field represents the changed value. Basically, it shows how it was before.
To	This field represents what is the new value for a given activity.
Field	This field represents which field was being changed.
UserId	This field represents the user who made the change.
CreatedDate	This field represents the date of the activity.

IssueRepositoryUser Table

This table is created for capturing issue repositories' users.

Id	This field represents the unique id of a given issue repository user. Our database assigns this id to a given issue repository user with FIFO order.
UserId	This field is coming from the user table which identifies the user.
IssueRepositoryId	This field represents the corresponding issue repository for a given issue repository user.
Username	This field represents the username of a given issue repository user.
UserEmail	This field represents the email of a given issue repository user.

User Table

This table is created for capturing user details. Especially it is useful when it comes to link users with social media. This table is the same table with social media database model.

Id	This field represents the unique id of a given user. Our database assigns this id to a given user with FIFO order.
Name	This field represents the name of a given user.
RelatedUserIds	This field represents the related user ids of a given user.

Reason	The field represents the reason why these users are related with each other.

Comments Table

This field is created for capturing comments for a given issue entry.

Id	This field represents the unique id of a given comment. Our database assigns this id to a given comment with FIFO order.
IssueId	This field derives from issues table. This is a unique identifier for an issue.
UserId	This field comes from the issue repository user table which is a unique identifier created with a given issue repository user.
Context	This field encapsulates the content of a given comment. In other words, we can define it as body of a given comment.

Priorities Table

This table is created for capturing priorities. Priorities can differ from project to project. Thus, we have created a table for capturing each projects' priority information.

Id	This field represents the unique id of a given priority. Our database assigns this id to a given priority with FIFO order.
IssueRepositoryId	This field represents the corresponding issue repository for a given priority.
PriorityName	This field represents the name of a given priority. As example: major, critical, and minor.
Description	This field represents the description of a given priority.

IssueRepository Table

This table is created for capturing the issue repository information.

Id	This field represents the unique id of a given issue repository. Our database assigns this id to a given issue repository with FIFO order.
ProjectId	This field derives from the project table which identifies the

	unique project id.
IssueRepositoryUrl	This field represents the issue repository URL of a given issue repository.
IssueRepositoryType	This field represents the issue repository type of a given issue repository.

Project Table

This table is created for capturing project information.

Id	This field represents the unique id of a given project. Our database assigns this id to a given project with FIFO order.
ProjectName	This field represents the project name of a given project.
ProjectUrl	This field represents the project URL of a given project.

Social Media Channels

SocialMediaEntries Table

This table is created for capturing information about the social media entries.

Id	This field represents the unique id of a given social media entry. Our database assigns this id to a given social media entry with FIFO order.
SocialMediaRepositoryId	This field derives from social media repository table which identifies the social media repository for a given social media entry.
SenderUserId	This field represents the author of a given social media entry.
OriginalEntryId	This field represents the original entry id derived from the social media entry of a given social media entry.
Context	This field represents the context of a given social media entry.
InResponseTo	This field is present, If a social media entries is a reply to a different social media entry then, we capture this information this field.

Receiver	This field represents the receiver of a given social media entry.
Subject	This field represents the subject of a given social media entry.
SentDate	This field represents the sent date by user of a given social media entry.
ReceivedDate	This field represents the received date by user of a given social media entry.
SeenDate	This field represents the seen date by user of a given social media entry.
Attachments	This field represents the attachments of a given social media entry.
Location	This field represents the location of a given social media entry.

SocialMediaEvents Table

This table is created for capturing social media events e.g. nickname change

Id	This field represents the unique id of a given social media entry. Our database assigns this id to a given social media entry with FIFO order.
SocialMediaRepositoryId	This field derives from social media repository table which identifies the social media repository for a given social media event.
SocialMediaUserId	This field derives from social media user table which identifies the social media user for a given social media event.
EventDate	This field represents the date of a given social media event.
EventType	This field represents the type of a given social media event.
Content	This field represents the content of a given social media event.

SocialMediaRepository Table

This table is created for capturing the social media repository information.

Id	This field represents the unique id of a given social media repository. Our database assigns this id to a given social media repository with FIFO order.
ProjectId	This field derives from the project table which identifies the unique project id.
RepositoryUrl	This field represents repository URL of a given social media repository
RepositoryType	This field represents repository type of a given social media repository

SocialMediaUser Table

This table is created for capturing the social media user information.

Id	This field represents the unique id of a given social media user. Our database assigns this id to a given social media user with FIFO order.
UserId	This field derives from user table which is an identifier of a user.
SocialMediaRepositoryId	This field derives from social media repository table which identifies the social media repository for a given social media user.
Username	This field represents the username of a given user.
UserEmail	This field represents the user email of a given user.

II. JIRA database schema

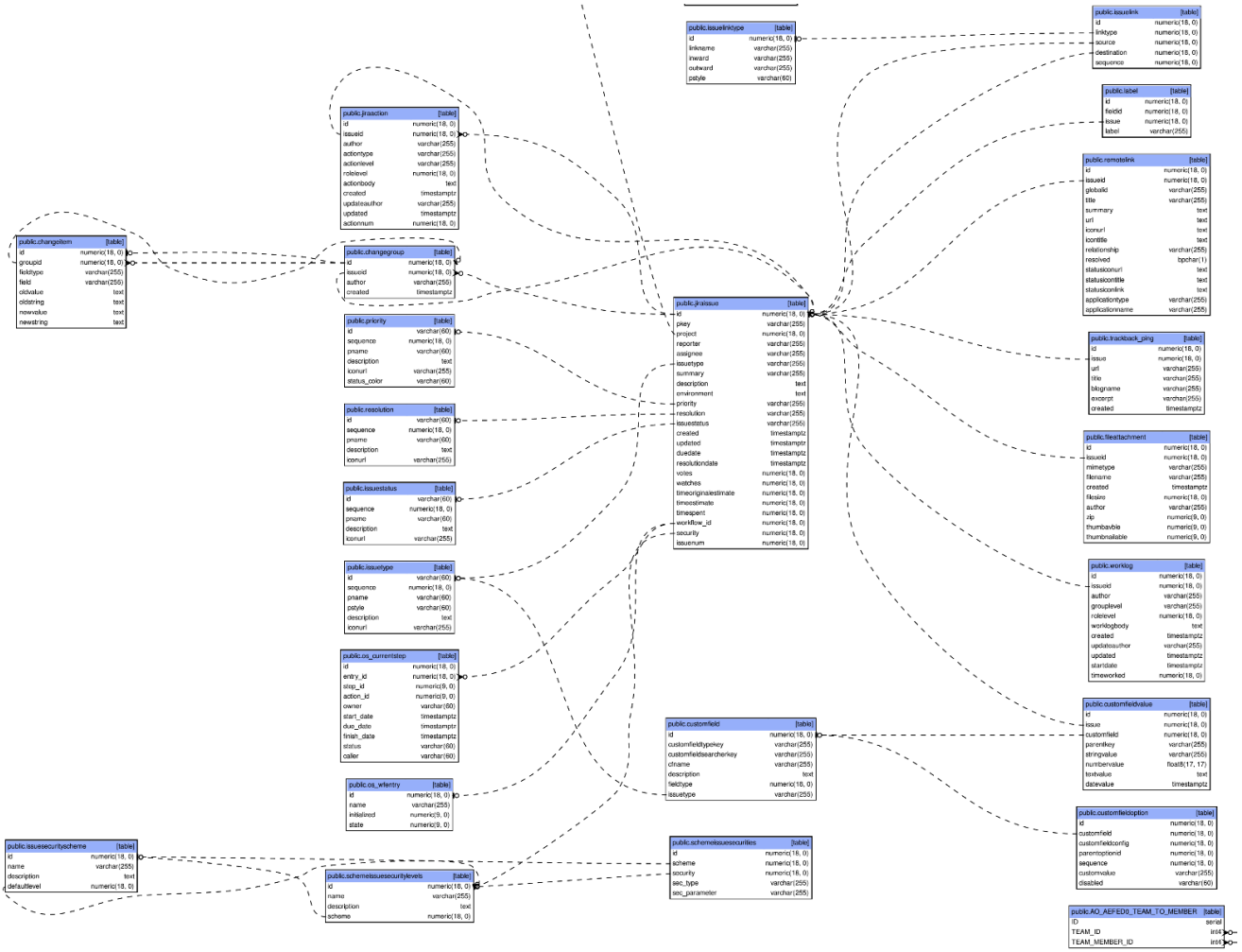

Figure 25. JIRA database schema [23]

[23] https://developer.atlassian.com/jiradev/files/4227160/JIRA61_db_schema.pdf

III. Bugzilla database schema

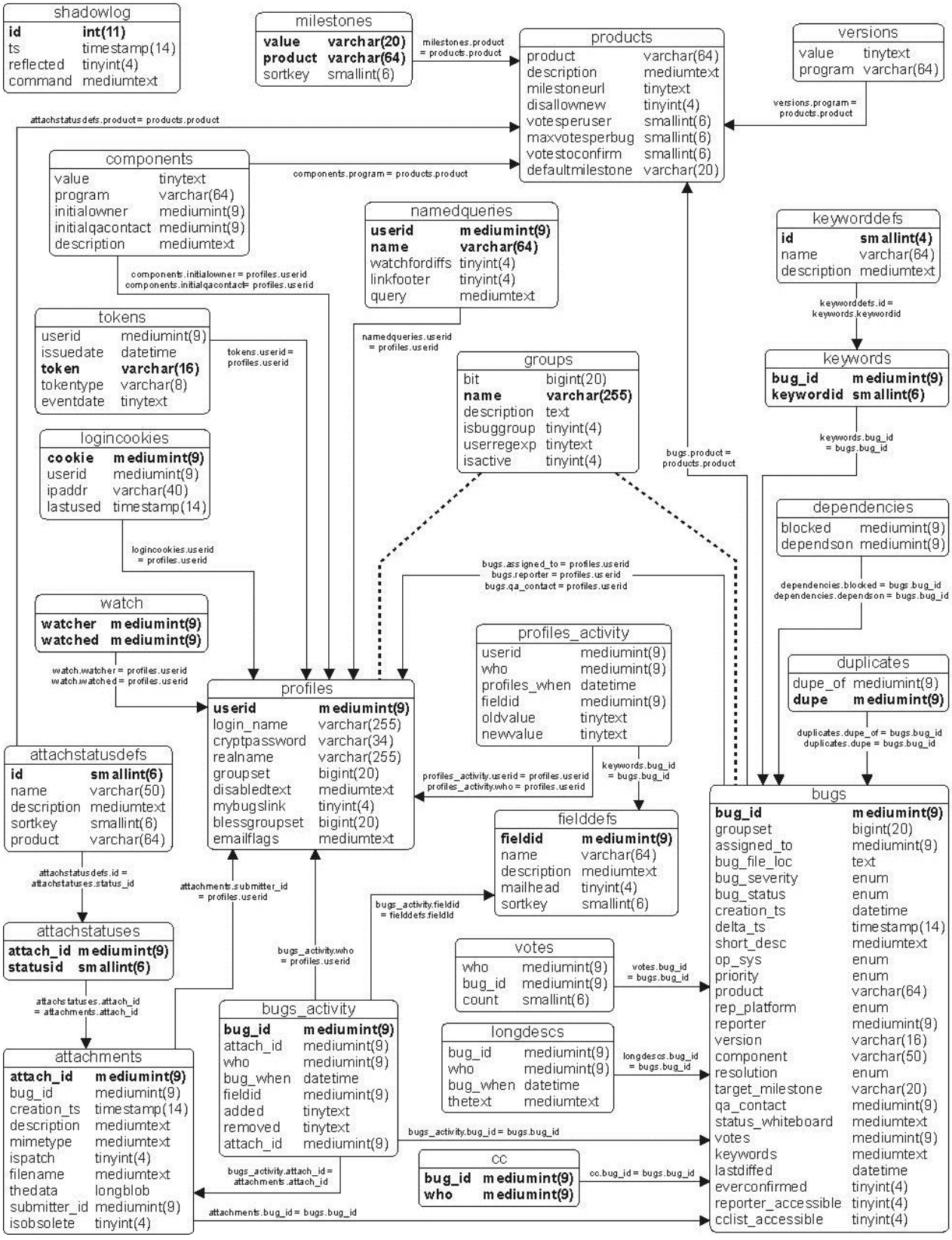

Figure 26. Bugzilla model chart [24]

[24] https://www.bugzilla.org/docs/2.16/html/dbschema.html

IV. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **İlgün İlgün** (date of birth: 01.03.1990),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Tools for software project data collection and integration,

supervised by Siim Karus,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **21.05.2015**