<table>
<thead>
<tr>
<th>Title</th>
<th>Large scale data integration of OSS repositories for automated soft and technical factors assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Iqbal, Muhammad Aftab</td>
</tr>
<tr>
<td>Publication Date</td>
<td>2015-04-22</td>
</tr>
<tr>
<td>Item record</td>
<td><a href="http://hdl.handle.net/10379/4963">http://hdl.handle.net/10379/4963</a></td>
</tr>
</tbody>
</table>
LARGE SCALE DATA INTEGRATION OF OSS REPOSITORIES FOR AUTOMATED SOFT AND TECHNICAL FACTORS ASSESSMENT

MUHAMMAD AFTAB IQBAL

Dissertation
submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy
April 2015

Supervisors: Dr. Michael Hausenblas and Prof. Dr. Stefan Decker

Internal Examiner: Dr. John Breslin

External Examiners: Prof. Dr. Asunción Gómez-Pérez and Dr. Jeff Z. Pan

Insight Centre for Data Analytics
National University of Ireland, Galway / Ollscoil na hÉireann, Gaillimh
ABSTRACT

As of today, software development does not revolve around a piece of source code but also around large volume of software project related information that exists in different software repositories hosting a software project. These software repositories produce a variety of software artifacts (i.e., source code, bugs, source control commit logs, emails, documentation etc.) during the whole software development lifecycle. Apart from the software project information that is distributed across different software repositories of a software project, software project related information is also distributed on the Web in heterogeneous open source software repositories. Examples of these open source software repositories are: collaborative infrastructure for software project development (i.e., code forges), social networking infrastructure (e.g., Twitter) to disseminate software project related information to a wider audience and statistical services that provides statistical information about software project development. Hence, we can say that information related to software projects are distributed on the Web. The information contained inside these heterogeneous software repositories is vital to software stakeholders for their day to day development needs. However, this information is not readily accessible due to the distributed nature and lack of integration among software repositories.

In this thesis, we propose to integrate software repositories by exploiting a Linked Data approach that allows an easy integration and identification of related information about software artifacts across heterogeneous software repositories. We start by describing our approach to publish and integrate software repositories (based on software artifacts) using Linked Data and show how the interlinked information can be delivered to software stakeholders in their development environments. Further, we present our approach to identify and interlink different and multiple IDs of a software developer, which he/she uses to interact with different software repositories of a software project. Moreover, we present some use case scenarios that can be realized by interlinking multiple IDs of a software developer.

With respect to hosting of software projects on publicly available development infrastructures (i.e., code forges), we propose to integrate different code forges based on metadata, similar software projects and software developers. We demonstrate the integration of software project and software developer related information across different code forges as well as relevant information that are available through statistical services. Further, we show that it enables software stakeholders to not only query statistical information about a software project as well as software developer but also allow them to keep track of the involvement of software developers in multiple software projects across different code forges.

In regards to the social aspects, we present evidence that software project and software developer related information also exists on social media channels. Based on our case study of the usage of Twitter by software developers, we motivate the integration of social media channels and software repositories. Finally, we exploit our linked datasets to investigate the evolving social dependencies and social relations among software developers over the period of time.
I declare that this thesis is composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Galway, Ireland, April 2015

Muhammad Aftab Iqbal
ACKNOWLEDGEMENTS

In the name of Allah, Most Gracious, Most Merciful

All praise and glory be to Allah Almighty, who gave me the strength, patience and His blessings to carry out the research work and produce this dissertation.

I would like to take this opportunity to express my special appreciation and thanks to my supervisors Michael and Stefan for their support, advice, guidance and useful critiques for this research. I would also like to thank Giovanni who gave me the internship opportunity at first place to come to Insight Centre for Data Analytics, NUI Galway. He believed in me during my internship period and therefore offered me a PhD opportunity. Special thanks to my examiners Asunción, Jeff and John for their critical review and their valuable feedback and discussion during the viva. I would also like to take this opportunity to thank all my colleagues in Insight Centre for Data Analytics, NUI Galway with whom I shared moments of fun, inspiring conversations and the many fruitful collaborations.

Finally, but perhaps most importantly, I would like to express my deepest gratitude to my parents, siblings and especially my beloved wife, Aisha, who sacrificed much of her life for me and was always there for me. Your prayers for me was what sustained me thus far.
CONTENTS

1 INTRODUCTION 1
1.1 Heterogeneous Software Project Data Sources  2
  1.1.1 Code Forges  3
  1.1.2 Social Media Channels  4
  1.1.3 Software Analytics Services  5
1.2 Importance of Integrating Distributed Software Project Related Information  8
  1.2.1 Integrating Intra-project Software Repositories  8
  1.2.2 Integrating Inter-project Software Repositories  8
  1.2.3 Integrating Social Media Channels  9
1.3 Research Problem  9
  1.3.1 Problems in Integrating Software Project Information  11
1.4 Requirements Elicitation  13
1.5 Publishing Software Project Information as Linked Data  14
  1.5.1 Understanding the Principles  15
  1.5.2 Understand your Data  15
  1.5.3 Choose URIs for Things in your Data  15
  1.5.4 Setup your Infrastructure  16
  1.5.5 Link to other Datasets  18
1.6 Research Hypotheses  18
1.7 Contribution  20
1.8 Impact  22
1.9 Thesis Outline  23

2 BACKGROUND AND STATE OF THE ART 25
2.1 The Semantic Web  25
  2.1.1 Resource Description Framework  25
  2.1.2 RDF Schema  26
  2.1.3 Web Ontology Language  27
  2.1.4 SPARQL  28
2.2 RDF Vocabularies  29
  2.2.1 SKOS  29
  2.2.2 FOAF  30
  2.2.3 SIOC  30
  2.2.4 DCMI  31
  2.2.5 MOAT  31
  2.2.6 BAETLE  32
  2.2.7 DOAP  32
2.3 Mining Software Repositories  33
  2.3.1 Semantic Web Based Software Data Integration  34
2.4 Social Aspects of Software Development  35
  2.4.1 Socio-Technical Dependencies among Software Developers  37

3 HARNESSING DISTRIBUTED SOFTWARE ARTIFACTS 39
3.1 Integrating Software Artifacts: Linked Data Approach  43
6.2.2 Evaluation .......................................................... 146
6.3 Conclusion ............................................................ 149

7 EXPLOITING LINKED DATASETS TO STUDY THE SOCIAL DYNAMICS 151
7.1 Identifying Social Dependencies and Social Interactions ................. 151
    7.1.1 Identifying Socially Dependent Software Developer Pairs ........... 152
    7.1.2 Identifying Socially Related Software Developer Pairs .............. 153
    7.1.3 Discussion ......................................................... 154
7.2 Studying Social Dependencies and Interactions of Developer Pairs ......... 156
    7.2.1 Social Dependencies of Software Developer Pairs .................. 156
    7.2.2 Social Interactions of Software Developer Pairs .................. 157
    7.2.3 Discussion ......................................................... 158
7.3 Correlation between Social Dependency and Interaction of Developers .... 159
    7.3.1 Correlation ......................................................... 161
7.4 Conclusion ............................................................ 162

8 CONCLUSION .......................................................... 163
8.1 Contributions ......................................................... 163
8.2 Lessons Learnt ......................................................... 165
8.3 Future Direction ....................................................... 166
    8.3.1 Standardised Initiatives ........................................ 166
    8.3.2 Software Development in the Cloud ................................ 167

A SOCIAL DYNAMICS OF SOFTWARE DEVELOPERS 189
    A.1 Socially Dependent Software Developer Pairs of Apache Projects ..... 189
    A.2 Socially Related Software Developer Pairs of Apache Projects ....... 194
    A.3 Social Dependencies and Social Interactions among Software Developer Pairs of Apache Projects .................................. 199

B QUERIES .............................................................. 209
C RDF SNIPPETS .......................................................... 215
In the software development process, both humans and software repositories are involved. Several phases undergo the software development process: starting with planning, followed by implementation, testing, documentation and finally the deployment and maintenance phase. During these phases, a variety of software repositories are being used by software developers in order to manage the software project. These software repositories produce a variety of software artifacts (e.g., source code, bugs, emails, requirements, plans etc.) during the complete software development lifecycle. Examples of these software repositories are [Hassan et al., 2005]:

1. **source control repositories** store latest changes to the source code as well as the software developers who have made those changes as development progresses;

2. **bug repositories** keep track of the software defects as well as the software developers who have reported and fixed those defects;

3. **archived communications** between software developers record rationale for decisions throughout the life of a software project.

Based on the different software repositories that are used by software developers to manage their development tasks, we can say that software development does not revolve around a piece of source code but also around large volume of software development-related information that exists in different software repositories, which are part of the software development process. Moreover, information related to software projects are also available on the Web in heterogeneous software repositories (e.g., code forges, social media channels, analytical services etc.). Software developers use these software repositories to manage and promote collaboration in distributed software development. These software repositories contain a wealth of valuable information about the history of a software project. As mentioned by [Cubranic et al., 2005], the different software artifacts contained within these software repositories implicitly forms a project memory.

The information about software artifacts are often required by software stakeholders (i.e., software developers, project managers, end-users etc.) on daily basis for their specific needs. For example, who has made latest changes to “Exception.java” and if those changes were made to fix any particular bug? In order to answer this, software stakeholders are required to look into the source control commit logs to find when “Exception.java” was changed, who made the changes and if it fixes any particular bug. Due to the lack of integration among software repositories based on similar or related software artifacts, information from different software repositories about software artifacts are not easily accessible. Therefore, software stakeholders have to manually look into each software repository in order to retrieve related information about a particular software artifact which is often time consuming.

To overcome these issues, we are required to develop a generic framework where information about a software project from heterogeneous software repositories can be collected and integrated.

---

We use the term *large scale* in this thesis to refer to the vast amount and different varieties of information relevant to a software project that exists in heterogeneous software repositories (e.g., mailing lists, bug repositories, source control repositories, social media channels, code forges etc.).
Therefore, in this thesis we aim to achieve this by exploiting a standard approach that allows an easy identification and integration of software project related information (based on similar software artifacts) from heterogeneous software repositories. Thus, we demonstrate how the standard approach that we propose can be applied to integrate various software repositories (bug repository, source code, source control repository & mailing list) hosting a software project. Later, we use the same approach to integrate software project related software repositories that are available on the Web (i.e., code forges, social media channels & analytical services) and show that information related to a particular software artifact can be queried across different software repositories easily and efficiently.

The remainder of this chapter is structured as follows:

- In Section 1.1, we highlight different data sources that are available on the Web and contains software project related information. We discuss the purpose and kind of information stored within those data sources. Further, we show through an example of a real world software project that information related to software projects are distributed on the Web within these heterogeneous data sources.

- Based on the fact that software project related information exists across different software repositories, we argue the importance of integrating software project related information within and across various software repositories in Section 1.2. This is followed by our research problem in Section 1.3, which is based on the information needs of software stakeholders and issues in integrating software project related information across different software repositories.

- We identify the key software artifacts that are contained within different software repositories in Section 1.4 and propose a Linked Data driven approach that allows an easy identification and integration of software artifacts across different software repositories.

- In Section 1.5, we outline the key steps in publishing software project related data as Linked Data by taking into account an example of a software project.

- Next, in Section 1.6, we outline our research hypotheses based on the research problem described in Section 1.3, followed by contribution of this thesis in Section 1.7. In Section 1.8, we mention various international events where parts of this thesis have been published and finally, outline the remainder of this thesis in Section 1.9.

1.1 HETEROGENEOUS SOFTWARE PROJECT DATA SOURCES

Apart from the software project related information that exists in different software repositories within enterprise settings, information related to software projects are also available and distributed on the Web in heterogeneous data sources\(^2\), i.e., code forges, analytical services and social media channels, as depicted in Figure 1.1.

The software project related data sources depicted in Figure 1.1 shows collaborative infrastructures for software development (i.e., code forges), social networking infrastructures to disseminate information related to software projects to a wider audience and infrastructures that provides statistical information about the history of software project development. In the following, we briefly

\(^2\) We use the term “data sources” in this chapter to take into account those sources that are not classified as software repositories (e.g., social media channels).
look into these heterogeneous data sources and further discuss the purpose of these data sources and the kind of information it contains about software projects.

1.1.1 Code Forges

Code forges are Web-based collaborative platforms supporting collaboration among software developers on the Web. Examples of code forges are: GitHub\(^3\), GoogleCode\(^4\) and SourceForge\(^5\). A code forge typically provides a set of software tools/repositories, which are required to manage and develop a software project. Code forges provide different kinds of features in order to attract the attention of software developers to host their software projects on their infrastructure. These features include managing project code, a place for the users to download the project releases, discussion forums, bug repositories, mailing lists etc. Therefore, instead of installing and hosting software repositories internally to manage a software project, software development teams often hosts their project on code forges, hence developing Open Source Software (OSS).

OSS development is a way for building, deploying and sustaining large software systems on global basis [Scacchi, 2007]. It has gained a lot of attraction from the public and the software engineering community over the past decade [Iqbal, 2014]. Thousands of OSS projects have been

---

\(^{3}\) https://github.com/
\(^{4}\) http://code.google.com/
\(^{5}\) http://sourceforge.net/
developed and are used by millions of end-users. The many OSS projects exist on the Web are driven by thousands of software developers geographically distributed and contributing to these software projects. The success of an OSS project is highly dependent on the infrastructure (i.e., source control repository, bug repository, documentation tool, mailing list, discussion forum etc.) provided by the code forges to the software developers and users in order to collaborate with each other [Shibuya and Tamai, 2009].

1.1.2 Social Media Channels

The social aspects of software development have been given increased attention by researchers over the past decade. Researchers have expanded how the communities of software stakeholders communicate, coordinate and collaborate with one another [Begel et al., 2013]. According to [Storey et al., 2010], the use of social networking have eliminated boundaries, which may be the blocker in the flow of information within and between organisations. The current generation of technology, also known as social media makes it easier for the users to share information with others in their social network [Begel et al., 2010]. In the following, we provide a brief summary of modern social media tools that are used by software developers to organize or disseminate software project related information to a wider community.

**MICROBLOGS**

Microblogging websites like Twitter\(^6\) enable users to share short text updates with a set of “followers” who have subscribed to receive the updates. Software projects are often found to adopt an identity on Twitter (e.g., Apache Solr/Lucene\(^7\)) in order to disseminate project related information (release announcement, major bug fixes etc.) or gather feedback/questions posted by the users. Moreover, software developers contributing to software projects also exists on Twitter. Quite often they discuss, debate or share experiences with others relevant to a software project using project related hashtags (e.g., #apache, #maven, #hadoop etc.).

**SOCIAL NETWORKING WEBSITES**

Social networking websites like Facebook and LinkedIn functions like an online community of users on the Web. Users of these websites share common interests, hobbies, work-related activities etc., with their friends or specific groups, which exists on these websites. Software projects are often found to adopt an identity on these social networking sites (e.g., MySQL\(^8\) on Facebook) so that they can gather the experiences of software users as well as share important software project related news/announcements with them. Moreover, software developers also exists on these social networking sites sharing work-related activities with co-workers and friends in their social circle.

**QUESTION ANSWER (Q&A) WEBSITES**

There are various tools that a software developer use to help manage his/her development tasks. However, little is known what a software developer needs to know about a code base or a component before making a change to it. There has been various studies on questions that are generally asked by software developers [Fritz and Murphy, 2010; Ko et al., 2007; LaToza and Myers, 2010; Letovsky, 1986; Sillito et al., 2006]. In contrast to these studies, the success of social media has

---

\(^6\) https://twitter.com/

\(^7\) https://twitter.com/SolrLucene

\(^8\) https://www.facebook.com/mysql
introduced new ways of exchanging knowledge between software developers via the Web. For example, Stack Overflow\(^9\) facilitates the exchange of software related knowledge between software developers. It is a popular Q&A website for software developers that focuses on technical questions related to programming [Begel et al., 2010]. The questions are aimed at a general audience that is not necessarily to be part of the same software project and the community anticipate questions by answering them as well as sharing their opinions/experiences through comments and ratings [Treude et al., 2011]. Answers on Q&A websites like Stack Overflow often becomes a substitute when the official documentation of a specific software or component is not available [Treude et al., 2011].

**BLOGS**

Blogs are discussion-oriented or informational websites where individuals share their opinions, thoughts and experiences with others. They are a series of articles on narrow topics with feedback in the form of comments from readers. Software developers often use blogs to share technical information about installing or configuring a particular software package/framework with a wider audience. Software companies frequently use blogs to share technical information about their products and services with their employees and customers, both internally and externally.

1.1.3 Software Analytics Services

Within the large volume of software project related data contained in either in-house software repositories or in code forges, there is hidden information about the quality of software, services and dynamics of software development. Various analytical approaches (e.g., data mining, machine learning, data visualization) could potentially be used to enable software stakeholders perform analysis on the data hidden within software repositories in order to obtain meaningful information about a particular software project [Buse and Zimmermann, 2010; Zhang et al., 2011].

There are a number of software tools, which are capable of monitoring and reporting different types of statistics/metrics in order to better support and manage a software project – such as, PROM [Sillitti et al., 2003] and Hackystat [Johnson et al., 2005]. For the open source community, Ohloh\(^{10}\) is developed to monitor an up-to-date development activity of OSS projects. Ohloh allows software developers to join (i.e., adopt an identity) and claim their contributions (in terms of source control commits) on existing software projects and also allows them to add software projects that are not yet on Ohloh, in order to assemble a complete profile of their OSS project contributions.

In this section, we have briefly discussed different data sources that are available on the Web and contains software project related information. Now, we take an example of a real world software project and provide evidence that information relevant to that software project is distributed across different data sources on the Web. We consider Apache Lucene\(^{11}\) project as an example in this chapter because of its open source nature. The reason of choosing an open source software project is that the information relevant to an open source project is discussed openly and available freely on the Web in contrast to proprietary softwares whose information is partially available on the Web. We have summarized different types of information relevant to Apache Lucene that are

\(^{9}\) http://stackoverflow.com/
\(^{10}\) http://www.ohloh.net
\(^{11}\) http://lucene.apache.org/
available via different data sources in Table 1.1. The table shows that information about Apache Lucene is not limited to its actual development activity but rather each data source listed in Table 1.1 serves a specific purpose and provides a different view on Apache Lucene project. For example, dissemination activity of Apache Lucene are carried out on microblogging websites like Twitter and Facebook. Various blogs are available on the Web that provide useful insights into Apache Lucene project, some of which are listed in Table 1.1. Additionally, specific technical questions/issues related to Apache Lucene are discussed on Q&A websites like Stack Overflow. Moreover, statistical information about the development activity of Apache Lucene is available via Ohloh. This shows that information about a software project is not limited to a single software repository but instead it is distributed on the Web in heterogeneous data sources. We have highlighted some data sources in Table 1.1, however, we believe that there are more available on the Web, which contains information related to Apache Lucene project.

Based on the heterogeneous data sources that are discussed briefly and the real world example of the existence of software project information in these data sources (cf. Table 1.1), we conclude that information relevant to a software project is distributed on the Web in heterogeneous data sources. This information is vital to different software stakeholders for their specific needs. However, this information is not readily accessible due to the distributed nature and lack of integration among these data sources. In the next section, we argue the importance of integrating these distributed and heterogeneous data sources through examples.
<table>
<thead>
<tr>
<th>Data Source</th>
<th>Source Type</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-house Software Repositories</td>
<td>source code</td>
<td><a href="http://www.apache.org/dyn/closer.cgi/lucene/java/4.6.0">http://www.apache.org/dyn/closer.cgi/lucene/java/4.6.0</a></td>
</tr>
<tr>
<td></td>
<td>source control</td>
<td><a href="http://svn.apache.org/repos/asf/lucene/">http://svn.apache.org/repos/asf/lucene/</a></td>
</tr>
<tr>
<td></td>
<td>bug tracking</td>
<td><a href="https://issues.apache.org/jira/browse/LUCENE">https://issues.apache.org/jira/browse/LUCENE</a></td>
</tr>
<tr>
<td></td>
<td>mailing list</td>
<td><a href="http://mail-archives.apache.org/mod_mbox/lucene-java-user/">http://mail-archives.apache.org/mod_mbox/lucene-java-user/</a></td>
</tr>
<tr>
<td></td>
<td>documentation</td>
<td><a href="http://lucene.apache.org/core/4_6_0/index.html">http://lucene.apache.org/core/4_6_0/index.html</a></td>
</tr>
<tr>
<td>Code Forges</td>
<td>GitHub</td>
<td><a href="https://github.com/apache/lucene-solr">https://github.com/apache/lucene-solr</a></td>
</tr>
<tr>
<td></td>
<td>SourceForge</td>
<td><a href="http://sourceforge.net/projects/lucene_mirror/">http://sourceforge.net/projects/lucene_mirror/</a></td>
</tr>
<tr>
<td>Social Media Channels</td>
<td>Microblogging</td>
<td><a href="https://twitter.com/solrlucene">https://twitter.com/solrlucene</a></td>
</tr>
<tr>
<td></td>
<td>Social Networking</td>
<td><a href="https://www.facebook.com/pages/Lucene/104032849634513">https://www.facebook.com/pages/Lucene/104032849634513</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="https://www.facebook.com/groups/183324115165204/">https://www.facebook.com/groups/183324115165204/</a></td>
</tr>
<tr>
<td></td>
<td>Q&amp;A community</td>
<td><a href="http://stackoverflow.com/questions/tagged/lucene">http://stackoverflow.com/questions/tagged/lucene</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://stackoverflow.com/questions/tagged/lucene-index">http://stackoverflow.com/questions/tagged/lucene-index</a></td>
</tr>
<tr>
<td></td>
<td>Blogs</td>
<td><a href="http://lucene.grantingersoll.com/">http://lucene.grantingersoll.com/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://searchhub.org/">http://searchhub.org/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://blog.sematext.com/">http://blog.sematext.com/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://www.sigram.com/blog/category/lucene/">http://www.sigram.com/blog/category/lucene/</a></td>
</tr>
<tr>
<td>Software Analytical Services</td>
<td>Ohloh</td>
<td><a href="http://www.ohloh.net/p/lucene">http://www.ohloh.net/p/lucene</a></td>
</tr>
</tbody>
</table>

Table 1.1: Information related to Apache Lucene project that is available on the Web in heterogeneous data sources.
1.2 IMPORTANCE OF INTEGRATING DISTRIBUTED SOFTWARE PROJECT RELATED INFORMATION

In the previous section, we discussed that software project related information is distributed on the Web in heterogeneous data sources. This information is often required by software stakeholders (i.e., software developers, managers, users of a software, companies looking to invest into a particular OSS project etc.) on daily basis for their specific needs. However, there is no seamless integration of software project related information across these data sources. Therefore, in this section we argue the importance of integrating software project related information across different software repositories as well as data sources (i.e., social media channels) with examples.

1.2.1 Integrating Intra-project Software Repositories

Software repositories hosting a particular software project contains valuable information that can be exploited to enhance the quality of software systems [Tappolet, 2008] and to guide decision processes in modern software development. It is worth mentioning that there is an implicit connection between information contained inside different software repositories. For example, a bug report may lead to discussions among software developers on the mailing list, change in the source code and additionally documentation needs to be updated. This may be reflected later in the configuration management systems. Additionally, a feature request may indirectly arise from a discussion among software developers or users of the software on the project mailing list.

Source code and bugs are often discussed on project mailing list or discussion forum. From the integration perspective, the information contained inside project mailing list can be analyzed to extract bug and source code related discussions, which can be linked to the actual bug information contained in a bug repository and to the actual source code. Such an integration will enable us to keep track of discussions among software developers that are happened on project mailing list and are relevant to a bug report or source code. Further, data in source control repositories can be analyzed to extract bug IDs and further link it to the actual bug information contained inside a bug repository in order to help software developers keeping track of: which source code files are committed together and what has been changed in source code files while fixing a particular bug report [Hassan, 2008].

1.2.2 Integrating Inter-project Software Repositories

Software projects quite often use third party libraries or components in order to increase productivity, saving time and reduce development cost. Components reuse focuses on reusing existing functionality and re-packaging code modules into new software projects. As Guillaume Rousseau mentioned in one of his blog post12:

Almost 3% of the open source projects ever created use at least one lib from Apache Commons!

Software reusability is a common practice in today’s software development process however, tracking software reusability at large scale has not been tackled yet. For example, it is still surpris-

---

12 https://fossbazaar.org/content/3-open-source-software-ever-created-use-apache-commons-libraries/
ingly difficult to tell how many software projects on GitHub uses Apache Logging\textsuperscript{13} functionality. Moreover, software projects are often listed on multiple code forges. The reason behind this might be the migration of software projects from one code forge to another over the period of time or the preference of one code forge over another by the software developers in order to develop software projects. Hence, the history of those software projects are distributed across multiple code forges. Having a connection between code forges based on similar software projects will support a complete view on the development history of a particular software project.

1.2.3 Integrating Social Media Channels

With the growing interest of social media usage; software developers and others interested quite often share experiences relevant to a software project on social media channels. The usage of hashtags or project related keywords in social messages (i.e., tweets, posts etc.) can be used to link software project related activity on the social media channels. This connection can be later exploited to address many use case scenarios, some of which are listed below:

- **Mining end-users response on the release or usage of a particular software project.** The variance and enormity of information that propagates through social media channels presents an opportunity to harness the data and build models on top to aggregate the opinions of the community related to a software project [Asur and Huberman, 2010];

- **Investigating the popularity of a particular software project.** The popularity of a software project on social media channels [de Vries et al., 2012] can be investigated by applying sentiment analysis on social messages [Agarwal et al., 2011; Pang and Lee, 2008].

In this section, we have argued with examples the importance of integrating software project related information contained within different software repositories hosting a particular software project as well as across different data sources (i.e., code forges and social media channels). In the next section, we look into different types of questions [Buse and Zimmermann, 2012; Ko et al., 2007; Sillito et al., 2006], which software stakeholders usually have in mind while exploring or developing a software project. In particular, we investigate the information needs from a software developer and company’s perspective and further outline the issues relevant to integrating software project related information across different software repositories.

1.3 **Research Problem**

Due to the different nature and purpose of each software repository, software project related information is not readily accessible. One has to manually look into a particular software repository in order to extract any relevant information which is often a daunting task. For example, a software developer is interested to know the primary development locations (i.e., code forges) where Apache Lucene is being partially developed except the development infrastructure provided by Apache Software Foundation\textsuperscript{14}. The software developer is forced to manually search all possible code forges to find traces of development activity relevant to Apache Lucene, which is a time consuming process. In regards to acquiring information about a particular software project,
software stakeholders often find it difficult in finding answers to questions like “What Happened” to more complex “How did it happen and why?” [Buse and Zimmermann, 2012].

OSS has been studied over the past decade from the organizational, institutional and economical aspects. Several studies have been carried out that describe how the OSS evolves over time [Godfrey and Tu, 2000; Nakakoji et al., 2002; Scacchi, 2003; Wang et al., 2007], what are the success factors [Crowston et al., 2003; Ghapanchi et al., 2011] and what are the motivational factors of the software developers [Hertel et al., 2003; Li et al., 2006; Ye and Kishida, 2003] that contributed to these project? Although these studies have been carried out in depth on core OSS projects, they still failed to provide an insight on what’s happening in a particular project? OSS projects are still considered to be a black box. Moreover, [Conklin, 2006] pointed out that it is difficult to obtain and abstract information from an OSS project in order to answer even simple questions like:

- how many software developers are working on a project?
- how big the community is surrounding a project?
- how many contributors are contributing to a project?
- what is the development ratio per software developer?
- is the project flourishing?

Above listed are some questions that are hidden deep inside the software repositories of an OSS project and usually have in the mind of software stakeholders. Further, the software project and developer related information is distributed on the Web in heterogeneous data islands (cf. Figure 1.1), making it difficult to keep track of the information. In the following, we elaborate on the different kinds of information relevant to a particular OSS project that is usually acquired by stakeholders.

SOFTWARE DEVELOPER’S PERSPECTIVE

There has been significant research into the information needs of software developers pertaining to the day-to-day software development tasks [Biehl et al., 2007; Ko et al., 2007; Sillito et al., 2006]. However, there hasn’t been much research on investigating the information needs of a software developer prior to joining a particular OSS project. OSS development is dynamic in nature. Software developers contribute to a project for a certain period of time and later leave the project or join another project of high interest. Hence, inviting software developers to the development team is an important aspect for the long term survival of an OSS project. Additionally, software developers often look to contribute to OSS projects that matches their interests. There are certain questions, a software developer usually have in mind prior to joining a particular OSS project. Some of which are listed below:

- what is currently happening in the project?
- what development has happened in the past?
- does the project looking for contributors?
- is there anything of my interest in the project?
- what can i contribute to the project?
With the rise in the adoption and success of OSS over the past decade, we have witnessed that OSS has reached a point where companies are seriously thinking about investing into OSS. With the involvement of companies, OSS will achieve sustained productivity and increased amounts of output will be produced [Capiluppi et al., 2012]. A report by Gartner Inc [gar, 2008] in 2008 reveals that 85% of companies surveyed currently using OSS in their enterprise and the rest are thinking of adopting OSS in the upcoming years. Companies are attracted towards open source rather than proprietary software because of quality, reliability and speed, not just cost savings. However, choosing the appropriate OSS for investment is still a challenge for companies because there are hundreds of open source communities developing systems. Moreover, OSS communities don’t have a marketing budget or don’t advertise their “products”, which makes it even more difficult for companies to know what potential OSS is available out there [Walli et al., 2005]. Additionally, companies look for answers to various questions before investing into a particular OSS project. Some of which are listed below:

- no. of active software developers/contributors;
- no. of issues reported and fixed monthly;
- no. of release downloads;
- popularity of the project in the OSS community;
- health of the OSS project and community surrounding it.

Despite the open access to the various software repositories hosting a particular OSS project as well as other relevant software repositories, finding answers to the questions listed above requires both time and effort. The reason is the lack of tools available to: navigate the data hidden inside different software repositories hosting an OSS project, integrate abstract level information from external software repositories to the actual software artifacts within software repositories, automate computation of various metrics etc. For example, information contained in Ohloh about the most commits made by a software developer on Apache Lucene project in a particular month must be interlinked with each individual commit made by that particular software developer in that particular month. This will allow software stakeholders to “drill down” from abstract level information (most commits) to the actual contributions (individual source control commits) made to Apache Lucene project by a software developer in a specific month. However, the software project related information contained inside different data sources (cf. Figure 1.1) are available in different formats and there is no common standard defined in order to bridge the connection between these distributed but logically related data sources, which is discussed in the following.

1.3.1 Problems in Integrating Software Project Information

In this section, we briefly discuss the issues of integrating software project related information across different software repositories.

INTEGRATING INTRA-PROJECT SOFTWARE REPOSITORIES

Software repositories hosting a software project, are produced by different vendors with a focus on solving a specific problem. For example, bug repositories are designed to help in keeping track
of software defects relevant to a software project. Source control repositories are designed to keep track and control on different versions of the source code, documentation, configuration files etc. Some of these software repositories are directly under the control of software developers, whilst others are shared among users and software developers, such as bug repository, documentation and project mailing lists.

Each of the software repositories serve a specific need and provides a unique view on the project. As mentioned by [Ramler and Wolfmaier, 2008], these individual views have to be merged in order to get a comprehensive overview. However, there are obstacles underway for seamless software repositories integration: data storage and representation — every software repository has its own implementation of storing and retrieving the data. Different software repositories serving a specific purpose also differ with each other in terms of their data storage and retrieval. For example, JIRA\(^{15}\) and BUGZILLA\(^{16}\) bug repositories have their own representation, implementation and schema for storing bugs related information. The same issue persists while integrating data across different software repositories. For example, bug repository store its data in relational database while mailing list archives implements the RFC 2822 standard [Resnick, 2001]. Moreover, some software repositories are desktop-based applications whilst others are Web-based applications, which makes it even more difficult to integrate information across software repositories.

**INTEGRATING INTER-PROJECT SOFTWARE REPOSITORIES**

As discussed in Section 1.1 and further supported with an example in Table 1.1, software project related information is distributed on the Web in heterogenous data sources. Each software repository tries to solve a specific problem and provides a unique view on the project. For example, OHloh provides statistical information about an OSS project but the actual day to day development activity of that OSS project can be found in software repositories hosted on a particular code forge.

As mentioned earlier, a software project sometimes developed at multiple code forges or migrates from one code forge to another over the period of time. Therefore, integrating software project related information across different code forges is required to enable a complete development history of an OSS project. However, different code forges available to date have their own database schema and represents data elements differently. Moreover, code forges sometimes use different terminologies to represent same things — such as, project topics, operating systems, programming languages etc.. For example, GOOGLECODE defines the term *Mac* to associate MACINTOSH as an operating system to the project’s metadata but SOURCEFORGE defines it as *OS X*. Although, both refer to the same operating system, different terms are used. Hence we require to define some kind of classification/mapping, which explicitly states that the two terms are semantically similar and belongs to the operating system family as discussed by [Iqbal et al., 2012a].

In this section, we have looked into the information needs of different stakeholders and outlined some questions from their perspectives. Further, we have discussed the problems we face while integrating software project related information across different software repositories. The overarching goal is to help software stakeholders move beyond information and towards useful insight. However, such a transition is not easy as it requires necessary integration of relevant information across different software repositories hosting a software project as well as across relevant data sources (e.g., social media channels) that are available on the Web.

15 [http://www.atlassian.com/software/jira/overview](http://www.atlassian.com/software/jira/overview)
1.4 Requirements Elicitation

Due to the sheer volume of data exists in different software repositories relevant to a software project, manually finding relevant information is time consuming and a major hurdle for software stakeholders to get insights into a particular OSS project. What is currently missing is the existence of a generic framework where the information about a software project from different software repositories can be collected, integrated and queried that allow software stakeholders to get an overall view of the software project. In order to collect and integrate information, the first and foremost thing is to identify the different possible software repositories and potential software artifacts contained within those software repositories. We have identified a list of potential data sources and key software artifacts contained within those data sources, which are listed in Table 1.2. The software artifacts outlined in Table 1.2 can be understood as entities, each of which provides valuable information. Moreover, the different software artifacts contained inside a particular software repository have relationships to each other. For example, consider the case of a software developer who makes a commit on a source control repository in order to fix a particular bug. The software developer actually establishes the relationship as an author of modified source code files and fixer of that particular bug. In this case, software developer, commit, source code and bug can be classified as entities and authoring of source code files and fixing a bug can be classified as relations between entities (i.e., software developer, source code and bug respectively).

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Source Type</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-house Software Repo.</td>
<td>source code</td>
<td>projects, software developers, classes, methods, software packages</td>
</tr>
<tr>
<td></td>
<td>source control</td>
<td>projects, software developers, source code, documentation, bugs, commits</td>
</tr>
<tr>
<td></td>
<td>bug tracking</td>
<td>projects, software developers, source code, bugs, commits</td>
</tr>
<tr>
<td></td>
<td>mailing list</td>
<td>projects, software developers, source code, bugs, commits</td>
</tr>
<tr>
<td></td>
<td>documentation</td>
<td>projects, software developers, source code</td>
</tr>
<tr>
<td>Code Forges</td>
<td>GrHub, Google-code, SourceForge etc.</td>
<td>projects, software developers, source code, documentation, bugs, commits, emails</td>
</tr>
<tr>
<td>Social Media Channels</td>
<td>Microblogging</td>
<td>projects, software developers, source code, bugs, commits</td>
</tr>
<tr>
<td></td>
<td>Social Networking</td>
<td>projects, software developers, source code, bugs</td>
</tr>
<tr>
<td></td>
<td>Q&amp;A community</td>
<td>projects, software developers, source code, bugs</td>
</tr>
<tr>
<td></td>
<td>Blogs</td>
<td>projects, software developers, source code, bugs</td>
</tr>
<tr>
<td>Software Services</td>
<td>Analytical</td>
<td>projects, software developers, commits</td>
</tr>
<tr>
<td></td>
<td>Ohloh</td>
<td>projects, software developers</td>
</tr>
</tbody>
</table>

Table 1.2: Different types of software artifacts found in data sources.
Based on Table 1.2, we can say that information about a particular software artifact (e.g., software project, software developer, source code etc.) is available through different software repositories. For example, Apache Lucene is hosted on the Apache Software Foundation development infrastructure and also on GitHub (URLs are shown in Table 1.1). Further, statistical information about Apache Lucene is available through software analytical service (e.g., Ohloh). Therefore, one can retrieve information about the day to day development activity of Apache Lucene, available via software repositories hosted on the Apache Software Foundation development infrastructure as well as GitHub and statistical information about Apache Lucene can be retrieved through Ohloh. However, each software repository conveys different kind of information about a particular software artifact. Therefore, we require to define a mechanism through which we can uniquely identify similar software artifacts (i.e., entities) extracted from different software repositories. For example, there is a need to define a mechanism that allows to trace software developer’s activity on social media channels and his/her development activity on different software projects hosted on multiple code forges by establishing some kind of connection between data sources based on similar software developer profile.

As described in detail in Section 1.1, software project related information is available freely on the Web in heterogeneous data sources and the Web is built on connecting related documents. However, we are not interested in connecting documents but rather connecting software related data (i.e., software artifacts), which is hidden inside different software repositories distributed on the Web. In order to uniquely identify entities (i.e., software artifacts) within software repositories and across different data sources on the Web, we are required to denote entities through globally unique identifiers. Moreover, we are required to define the association between entities by describing their relationships explicitly. For example, “Mark” <isDeveloperOf> “Exception.java”. This association tells that a software developer named “Mark” is a developer of source code file named “Exception.java”. Given that the information available through different data sources are represented in different formats, therefore, there is a need to adopt a standard approach (see Section 1.5) that allows us to easily describe and integrate software project related entities contained inside distributed and heterogeneous data sources. In particular, we require methods and technologies to extract the information contained inside various data sources (e.g., see Table 1.1) and publish it on the Web in a structured format for integration purposes.

One of the best practices for publishing and connecting structured data on the Web is Linked Data. Key technologies that support Linked Data are HTTP URIs (a generic means to identify and retrieving descriptions of entities in software repositories) and RDF (a graph-based data model with which to describe structured information about software artifacts and define relationships (i.e., links) between software artifacts contained in software repositories). Given the heterogeneous nature of software repositories and the power of RDF to describe relations between different entities, we highlight the fact that by utilizing Linked Data approach, a Web-scale integration of software project related information is hence made possible. Therefore, in the next section we present a Linked Data driven approach to publish software project information from the integration perspective.

### 1.5 Publishing Software Project Information as Linked Data

Linked Data is about publishing structured data on the Web and further connecting it to other related data that wasn’t previously linked. In 2006, [Berners-Lee, 2006] introduced the “Linked Data” design principles to create a Web of interlinked data. There have been various case studies
and literature published highlighting best practices of publishing data on the Web as Linked Data. Among them, [Heath et al., 2008] outlined 5 steps of publishing Linked Data on the Web. In the following, we look deeply into each of these 5 steps by taking into account an example of a software project whose information exists in heterogeneous data sources.

1.5.1 Understanding the Principles

The first step of publishing data as Linked Data is the basic understanding of Linked Data principles which are:

**LDP1** Use URIs as names to identify things;

**LDP2** Use HTTP URIs so that it can be dereferenced by software agents;

**LDP3** Provide useful information in RDF [Klyne et al., 2004] upon requests;

**LDP4** Include links to external dereferenceable URIs.

1.5.2 Understand your Data

The second step in publishing Linked Data is to understand the underlying data. As our focus in this thesis is on software project related information, therefore it is important to identify/highlight the key things (i.e., entities) present in heterogeneous data sources. We have already summarized the key things (i.e., software artifacts) found in different data sources in Table 1.2. Moreover, we observe in Table 1.2 that same type of software project related information (i.e., software artifacts) is available through different data sources. In order to interlink related piece of information across different data sources, we are required to assign URIs as names (first principle of Linked Data) to these software artifacts, which is discussed next.

1.5.3 Choose URIs for Things in your Data

The third step is to choose URIs as names for things. We choose URIs for different software artifacts that are found in heterogeneous data sources. Adhering to the Linked Data principles, we use HTTP URIs so that useful information about a particular software artifact can be provided upon request. The primary function of a URI is served through its unique association with a specific piece of information and the longevity of that association [Juty et al., 2012]. As a software repository usually contains information about various software artifacts (cf. Table 1.2), therefore, we devise different URI patterns for different software artifacts. Moreover, it is likely that a software artifact exists across different software repositories. For example, “Mark” is a software developer who is contributing to different software projects on Github and SourceForge. Therefore, we need to distinguish between his development activities on different code forges. Hence, we can not devise one URI for “Mark” in this case. One approach could be that we use code forge name as part of a URI such as: “/github/mark” and “/sourceforge/mark”. This way we will have different URIs for “Mark” i.e., one URI per code forge. Later, we can define the association between both URIs stating that it refers to the same software developer, as we discuss later in Section 1.5.5.

Based on Table 1.2, we have outlined exemplary URI patterns in Table 1.3 as a reference for data publishers in order to help them in defining their URIs for different software artifacts. The
domain name used in URI patterns (i.e., example.org) is for illustration purposes only and must be replaced by the domain name of a data publisher. For further details on the design of URI patterns, we refer the readers to [Berners-Lee et al., 2005].

As mentioned earlier, a variety of software repositories are used to manage a software project and each software repository serves a specific purpose. Therefore, we use software repository name along with the project name in the URI (e.g., /project-name/svn/, /project-name/bug/ etc.) in order to differentiate between the information coming from different software repositories relevant to a particular software project. For example, a bug and source control commit URI for Apache Lucene project may look like: http://example.org/lucene/bug/1242 and http://example.org/lucene/svn/216.

Code forges host thousands of software projects. Therefore, it is likely that different software projects with similar names are hosted on these code forges. Hence, we use code forge name in the URI in order to differentiate software projects (having similar names) from each other. For example, a software project URI on GitHub and SourceForge may look like: http://example.org/github/lucene and http://example.org/sourceforge/lucene. Moreover, the software repository URIs of each software project hosted on a code forge will follow the same principles as described previously.

URIs for social media channels and analytical services are build based on software project and software developer names. For example, a software developer URI on Twitter and Ohloh may look like: http://example.org/twitter/brettporter and http://example.org/ohloh/lucene/brett.

1.5.4 Setup your Infrastructure

The fourth step is to setup the infrastructure so that information can be represented in a format asked by either a machine or human. For example, serving a proper HTML representation along with a RDF representation of a resource helps human to figure out what a URI refers to. This can be achieved through a mechanism called content negotiation. An example of content negotiation can be seen in Figure 1.2, which is taken from [Bizer et al., 2007].

![Figure 1.2: An example of dereferencing a HTTP URI using content negotiation (picture taken from [Bizer et al., 2007]).](image-url)
<table>
<thead>
<tr>
<th>Data Source</th>
<th>Source Type</th>
<th>Example URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-house Software Repositories</td>
<td>source code</td>
<td><a href="http://example.org/%7Bproject-name%7D/java/Exception">http://example.org/{project-name}/java/Exception</a></td>
</tr>
<tr>
<td></td>
<td>source control</td>
<td><a href="http://example.org/%7Bproject-name%7D/svn/216">http://example.org/{project-name}/svn/216</a></td>
</tr>
<tr>
<td></td>
<td>bug tracking</td>
<td><a href="http://example.org/%7Bproject-name%7D/bug/1242">http://example.org/{project-name}/bug/1242</a></td>
</tr>
<tr>
<td></td>
<td>mailing list</td>
<td><a href="http://example.org/%7Bproject-name%7D/email/5bece89c957">http://example.org/{project-name}/email/5bece89c957</a></td>
</tr>
<tr>
<td></td>
<td>documentation</td>
<td><a href="http://example.org/%7Bproject-name%7D/doc/specification">http://example.org/{project-name}/doc/specification</a></td>
</tr>
<tr>
<td>Code Forges</td>
<td>GitHub, Googlecode, SourceForge etc.</td>
<td><a href="http://example.org/%7Bcodeforge-name%7D/%7Bproject-name%7D/java/Printer">http://example.org/{codeforge-name}/{project-name}/java/Printer</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/%7Bcodeforge-name%7D/%7Bproject-name%7D/svn/417">http://example.org/{codeforge-name}/{project-name}/svn/417</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/%7Bcodeforge-name%7D/%7Bproject-name%7D/svn/417">http://example.org/{codeforge-name}/{project-name}/svn/417</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/%7Bcodeforge-name%7D/%7Bproject-name%7D/bug/3422">http://example.org/{codeforge-name}/{project-name}/bug/3422</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/%7Bcodeforge-name%7D/%7Bproject-name%7D/email/5hg42q2d">http://example.org/{codeforge-name}/{project-name}/email/5hg42q2d</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/%7Bcodeforge-name%7D/%7Bproject-name%7D/doc/specification">http://example.org/{codeforge-name}/{project-name}/doc/specification</a></td>
</tr>
<tr>
<td>Social Media Channels</td>
<td>Microblogging</td>
<td><a href="http://example.org/twitter/%7Bdeveloper-name%7D">http://example.org/twitter/{developer-name}</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/twitter/%7Bdeveloper-name%7D/status/984323">http://example.org/twitter/{developer-name}/status/984323</a></td>
</tr>
<tr>
<td></td>
<td>Social Networking</td>
<td><a href="http://example.org/facebook/%7Bdeveloper-name%7D">http://example.org/facebook/{developer-name}</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/facebook/%7Bdeveloper-name%7D/status/294d3323w">http://example.org/facebook/{developer-name}/status/294d3323w</a></td>
</tr>
<tr>
<td></td>
<td>Q&amp;A community</td>
<td><a href="http://example.org/stackoverflow/%7Bdeveloper-name%7D">http://example.org/stackoverflow/{developer-name}</a></td>
</tr>
<tr>
<td></td>
<td>Blogs</td>
<td><a href="http://example.org/blog/%7Bdeveloper-name%7D">http://example.org/blog/{developer-name}</a></td>
</tr>
<tr>
<td>Software Analytical Services</td>
<td>Ohloh</td>
<td><a href="http://example.org/ohloh/%7Bproject-name%7D">http://example.org/ohloh/{project-name}</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://example.org/ohloh/%7Bproject-name%7D/(developer-name)">http://example.org/ohloh/{project-name}/(developer-name)</a></td>
</tr>
</tbody>
</table>

Table 1.3: Reference implementation of URI patterns for different software artifacts found in each data source.
1.5.5 Link to other Datasets

One of the primary reasons of publishing Linked Data is the creation of incoming and outgoing links among related data as they are the glue that connects data islands into a global, interconnected data space [Heath and Bizer, 2011]. The links between URIs of different things can be achieved by using various popular predicates, e.g., owl:sameAs, rdfs:seeAlso, foaf:topic etc. In the scope of this thesis, interlinking is about linking software project related information across different data sources based on logically related software artifacts as shown in Figure 1.3.

![Figure 1.3: Snapshot of different data sources that contains information about Apache Lucene and are integrated to each other through owl:sameAs links.](image)

**Figure 1.3** shows different data sources containing information about Apache Lucene in the form of graphs. These graphs are further connected to each other using owl:sameAs type links. Such an interconnected data space enables software stakeholder to query information of Apache Lucene from a variety of data sources to serve his needs.

1.6 RESEARCH HYPOTHESES

Software repositories that are used to manage an OSS project can be understood as heterogeneous, interconnected datasets (of implicit nature, e.g., mentioning of source code or a bug in the email), conveying information about the software project and the humans involved. However, the inter-connections among these software repositories are typically not explicit. For example, while working on a particular piece of code, a software developer might be interested in other information relevant to the task in hand. He might be interested to know who last updated the source code file
he is currently working on, what were the discussions exchanged among the software developers regarding the source code changes, what other source code files are dependent on this change etc. Quite often software developers seek such information from expert software developers but in distributed software development and specially in OSS development settings, it is quite difficult to gather such information, for example, because of the unavailability of expert software developers due to time zone differences. Hence, software developers are required to manually trace discussions on the mailing list, search for bugs on the bug repository, and extract the log information from the source control repository. Supporting integration between software repositories enable software stakeholders to easily find the information they are looking for and therefore, spend less time traversing through the software repositories. In addition to that it could serve as a single point of entry to measure various statistical metrics about the development of an OSS project, which can be exploited to assess the health of an OSS project.

Thus, our hypothesis can be summarized as follows:

Integration of heterogeneous and distributed software project related information will enable a qualitative and quantitative view of an OSS project and allows to assess various factors automatically and efficiently.

Assuming that the software repositories are integrated, one can compute various factors [Boehm, 1981; Wagner and Ruhe, 2008] in order to address different use case scenarios. These factors can be broadly classified into different categories [Sudhakar et al., 2011]: technical, non-technical, social, organisational and environmental. Examples of some factors are [Wagner and Ruhe, 2008]: communication culture of software developers, bugs reporting and fixing turnover, development ratio per software developer etc. The metrics computed individually for different factors based on the data contained inside different software repositories can be aggregated and represented as a single metric value as shown in Figure 1.4. Providing support to compute an aggregated metric value (cf. Figure 1.4) for an OSS project enables software stakeholders to quickly assess the overall health of an OSS project. Further, they will be able to decompose the aggregated metric value into individual metric values (computed based on different factors) in order to get more detailed information about each computed metric value.

In more detail, we investigate and validate the following hypotheses:

[H1] Integrating software project and software developer related information across OSS repositories and social media channels enable a comprehensive overview on the development of an OSS project.

[H2] We can efficiently query integrated software data and deliver related information about a particular software development task as well as software developers activities within and across different OSS projects automatically and efficiently.

[H3] We can efficiently query integrated software data to assess various technical and non-technical factors, such as:
- software developers development ratio;
- software developers development activities.

[H4] We can efficiently query integrated software data to assess social factors, such as:
- social dependencies and relations among software developers;
The answers to the research hypotheses outlined above would provide us a methodology that goes:

- from the collection of software project and software developer related data from heterogeneous software repositories;
- to the semantic enrichment of this data using RDF;
- to the aggregation of this data by inserting semantic links between similar or related software artifacts;
- and finally, querying integrated data to validate our research hypotheses and to address different use case scenarios.

1.7 CONTRIBUTION

This thesis aims at applying a standard approach i.e., Linked Data, in order to integrate the information contained inside different software repositories so that it enables stakeholders to assess various factors of an OSS project in an efficient and automated manner.

- We present LD2SD (Linked Data Driven Software Development), a Linked Data based approach to turn software artifacts such as data from mailing list, source code, source control and bug repositories into Linked Data, hence allowing uniform querying and browsing facilities. In addition, application of LD2SD on a reference software project is shown. **Contributes to H1 and addressed in Chapter 3.**

- In order to enable software developers to make effective use of interlinked dataset (based on LD2SD approach), we develop an interface that software developers can use as a plugin
in their development environment, such as Eclipse IDE in order to find related information about source code, which may exist in a bug repository, source control commit logs or mailing list archives. With respect to effectiveness, we evaluate our Linked Data based approach against traditional ways (manual approach) of finding relevant information from different software repositories by conducting an end-user evaluation. *Contributes to H2 and addressed in Chapter 3.*

- In order to identify and interlink multiple IDs of a software developer, we evaluate existing approaches and discover that they are not sufficient enough for the task at hand. Therefore, we propose our approach to identify and interlink multiple IDs of a software developer that are contained in different software repositories of a software project as well as across software repositories of different software projects. Further, we show that interlinking multiple IDs of a software developer helps in keeping track of development activity of software developers, which is distributed within different software repositories of a software project as well as across different software projects. *Contributes to H1, H2 & H3 and addressed in Chapter 4.*

- We then extend upon the work of interlinking software developer IDs by taking into consideration other relevant open source software repositories that are available on the Web. We propose to integrate software developer related information (by interlinking their respective IDs), which are found in software repositories of a software project, code forges (e.g., GitHub, SourceForge etc.) and software development statistical services, such as OHLH. Further, we show with query examples that such an integration enables software stakeholders to not only query facts that are hidden deep inside the software repositories but also allow them to query statistical information as well as development activity of a software developer in different software projects across multiple code forges. Moreover, we propose to integrate code forges based on metadata and similar projects. *Contributes to H1 & H3 and addressed in Chapter 5.*

- We study the usage of social media channels by software developers through harvesting their software project related activities on Twitter. We introduce a new dimension to the social aspects of software developers by taking into account non-traditional communication channels (i.e., Twitter, Facebook etc.) and study the social interaction behavior of software developers with each other across different communication channels. *Contributes to H1 & H4 and addressed in Chapter 6.*

- We present an overview of the social dependencies and social interactions among software developers over the period of time, by exploiting our interconnected datasets. We show that social dependencies among software developers can be easily queried and compared against their social interactions in order to identify the socio-technical dependency conflicts. *Contributes to H4 and addressed in Chapter 7.*
1.8 IMPACT

Various parts of this thesis have been published in international workshops, conferences and journals.

• We presented our fundamental approach to integrate software repositories based on Linked Data principles at the Software Engineering and Knowledge Engineering (SEKE) Conference [Iqbal et al., 2009b]. We developed an interface that allows software developers to consume the interlinked information from different software repositories and published a user study to evaluate the effectiveness of our approach against manual approach of finding relevant information from different software repositories at Semantic Web Enabled Software Engineering (SWSE) Workshop [Iqbal et al., 2009a];

• We evaluated existing approaches on identifying multiple IDs of a software developer that exists in a project mailing list and published our findings at the Knowledge Discovery, Data Mining and Machine Learning (KDML) workshop [Iqbal and Karnstedt, 2010]. In later years, we developed our own approach to identify and also interlink multiple IDs of a software developer based on Linked Data principles, compared the results of our approach against existing approaches and published our findings at ISRN Software Engineering Journal [Iqbal and Hausenblas, 2013];

• We then extend upon the work of integrating software project and software developer related information at large scale and published a technical report [Iqbal et al., 2012a], where we discuss the issues in integrating data from multiple code forges and propose to model metadata of code forges using RDF vocabularies for integration purposes.

• We further worked on integrating software developer related information (by interlinking software developer IDs) across open source repositories (i.e., code forges, statistical services) and published our findings at the Information Reuse and Integration (IRI) Conference [Iqbal and Hausenblas, 2012].

• We studied the usage of social media channels (i.e., Twitter) by software developers and carried out an experiment to understand the communicational behavior of software developers across different communication channels (i.e., project mailing list and Twitter). We published our findings at Software Engineering and Knowledge (SEKE) Conference [Iqbal et al., 2013].

• With respect to the dynamic nature of open source software community, we investigated the contributions made by those project members who had attained the role of a software developer in a software project. Further, we studied the significance of non-developers contributions to a software project and investigated if and to what extent they play a major role in the long term survival of an open source project. We published our findings at ISRN Software Engineering Journal [Iqbal, 2014].

• Finally, as part of future research directions, we published a technical report [Iqbal et al., 2012b], in which we reviewed and compared state of the art cloud-based software development environments. Moreover, we outlined a roadmap of cloud-based software development by identifying challenges and opportunities and proposed a set of requirements for the next generation of cloud-based development tools.
1.9 **Thesis Outline**

The remainder of the thesis is structured as follows:

**Chapter 2** introduces core Semantic Web concepts, such as RDF, OWL, SPARQL and provides a brief overview of different RDF vocabularies that are used in this thesis to represent software artifacts as RDF. Moreover, we briefly explain state-of-the-art on software repositories integration as well as social aspects of software development;

**Chapter 3** introduces our methodology to transform data from different software repositories into Linked Data. We further evaluate the effectiveness of our approach against traditional ways of finding relevant information from different software repositories by carrying out a user study;

**Chapter 4** focuses on identifying and interlinking different and multiple IDs of a software developer, which he/she uses while interacting with different software repositories of a software project;

**Chapter 5** provide details on integrating open source repositories based on software project metadata, similar software projects and software developers using Linked Data approach and further argue the benefit of integration through examples;

**Chapter 6** highlights the need of integrating software repositories and social media channels. We further study the behavior of software developers on social media channels by harvesting their project related activities;

**Chapter 7** identifies the social dynamics of software developers by depicting an overview of evolving social and technical dependencies among software developers working on a software project, by exploiting Linked Data approach;

**Chapter 8** provides discussion on the work, future research directions and concludes this thesis.
BACKGROUND AND STATE OF THE ART

In this chapter, we provide background knowledge by introducing the concepts of Semantic Web along with some RDF vocabularies that are used in this thesis. Moreover, we present the state of the art on mining software repositories and introduces the social aspects of software development.

2.1 THE SEMANTIC WEB

The idea of Semantic Web is to extend the traditional Web into a machine-readable Web, proposed by Berners-Lee [Berners-Lee, 1998] due to the shortcomings of the traditional Web with its vast amount of unstructured information. The ultimate goal of Semantic Web is to transform the human readable “Web of Documents” into a “Web of Data”\(^1\), such that machines can process the information and accomplish a wide variety of tasks, which otherwise have to be performed manually by users. The Semantic Web can also be considered as one giant global database, containing structured information that can be queried. In contrast to traditional databases, the information does not necessarily conform to a single schema and therefore can be heterogeneous. In order to enable machine-readable and machine-processable Web, the Semantic Web relies on two main fundamental technologies:

- a common model definition to represent and define associations/assertions between Web resources, which is made possible through “Resource Description Framework (RDF)”.
- formal vocabularies designed to represent the semantics of Web resources, which is made possible through “RDF Schema (RDFS)” and “Web Ontology Language (OWL)”.

2.1.1 Resource Description Framework

Resource Description Framework (RDF) was the first major step towards realising the machine-readable and machine-processable Web when it became the W3C Recommendation in 1999 [Lassila and Swick, 1999]. It is the fundamental data model of the Semantic Web and is generally used to describe meta data and to model information about any resource through RDF statements. Most importantly, it provides a standard mechanism of expressing information such that it can be exchanged easily between RDF-aware agents [Manola et al., 2004].

An RDF statement is a tuple that consist of three elements, a subject, a predicate and an object; also called a triple (a statement). A collection of RDF statements produces a directed graph where the subjects and objects are connected to each other through directed arrows and labels on the arrows represents predicates. An RDF resource can be anything with discernible identity, which can be identified by a URI, blank node\(^2\) or simply a string value (also called literal). A blank node is an unnamed resource, which is not identified with a URI or a literal and is expressed using the

\(^1\) “The Semantic Web is a Web of Data”, as deﬁned by W3C in http://www.w3.org/2001/sw/

\(^2\) http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-blank-nodes
notation _:name. In an RDF triple, subjects can be either URIs or blank nodes, predicates must be URIs and objects can be either URIs, blank nodes or literal values.

As mentioned above, RDF statements can be specified by means of triples and therefore, expressed as tuples of the form:

\[(\text{subject}, \text{predicate}, \text{object})\]

Let’s consider a sentence as an example: “Aftab knows Michael”. It can be represented as an RDF statement, where subject will be Aftab, predicate will be knows and object will be Michael. Naming is an important part of RDF. User-defined resources are thus optionally named using a URI, however, unnamed resources are usually represented as blank nodes. Most RDF syntaxes allow to use Compact URI (CURIE) [Birbeck and McCarron, 2009] names of the form prefix:reference to denote URIs. Considering our example, let’s assume that we have a namespace http://example.com/ abbreviated with example: (QName) for all three elements (i.e., subject, predicate and object). Thus, the above RDF statement can be illustrated in graph, as shown in Figure 2.1.

Figure 2.1: An exemplary RDF statement representing “Aftab knows Michael”.

Besides the graph illustration, RDF data can be serialised as RDF/XML [Swartz, 2004] and Terse RDF Triple Language (Turtle) [Beckett and Berners-Lee, 2008]. Each serialisation format has its own advantages and disadvantages depending on the use case. However, Turtle is seen as more human friendly in contrast to RDF/XML. Therefore, we use Turtle syntax throughout this thesis.

In Turtle syntax, RDF statements are written as whitespace separated triples. Moreover, angle brackets (<>) are used to denote URIs and quotes (“”) are used to denote literals. For example, Listing 2.1 shows two RDF statements about a person Aftab in Turtle syntax. The first statement tells that example:Aftab is a person (represented through foaf:Person) and second statement tells us that example:Aftab knows example:Michael.

```
1 @prefix example: <http://example.com/> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
4 example:Aftab rdf:type foaf:Person;
  foaf:knows example:Michael .
```

Listing 2.1: Turtle representation of RDF statements. We have also used 2 popular vocabularies in this example rather than only using “example” namespace.

2.1.2 RDF Schema

The RDF graph model does not provide any means to describe domain-specific classes and properties in order to represent information of resources in the form statements. However, it can be
described using RDF Vocabulary Description Language, referred to here as RDF Schema (RDFS). RDF Schema does not provide a vocabulary to define domain-specific classes and properties but provides a mechanism to define them and to indicate which classes and properties can be used together. In early 2004, the modern RDF specification became a W3C Recommendation [Brickley and Guha, 2004]. RDFS extends RDF semantics through four key terms, namely rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range.

rdfs:subClassOf allows to define the hierarchies between classes and therefore, allows to state that the instances of one class are also instances of another class. For example,

```
(example:Employee, rdfs:subClassOf, example:Person)
```

This states that all instances of example:Employee are also instances of example:Person. Similar to rdfs:subClassOf, rdfs:subPropertyOf describes organisations of properties between hierarchies. For example,

```
(example:technician, rdfs:subPropertyOf, example:worksFor)
```

This states that all resources related by example:technician are also related by example:worksFor. The last two terms (rdfs:domain and rdfs:range) are used to state that the resource for a given property is a member of a given class or the object value for a property is a member of a given class, respectively. For example,

```
(example:technician, rdfs:domain, example:Employee)
```

and

```
(example:technician, rdfs:range, example:Organisation)
```

RDFS also defines several other terms such as, rdfs:comment, rdfs:label, rdfs:seeAlso etc.

### 2.1.3 Web Ontology Language

The first thing required for the Semantic Web is to have an ontology language that can be used to formally describe the terminologies, their meanings and the relationships between those terminologies in Web documents. Therefore, Web Ontology language (OWL) originate from previous works - such as DAML [Hendler and McGuinness, 2000], OIL [Fensel et al., 2001] and DAML+OIL [McGuinness et al., 2002] and became a W3C Recommendation [McGuinness and van Harmelen, 2004] in 2004 after W3C began working on a new ontology language within the OWL working group in 2001. OWL extend upon RDFS with more expressive semantics for domains and is designed to be used when the information needs to be processed by machines instead of just presenting it to humans.

OWL consists of three sublanguages, that has different levels of expressivity (description of these sublanguages are taken from [McGuinness and van Harmelen, 2004]):

- **OWL Lite** – supports those users primarily needing a classification hierarchy and simple constraints.
• **OWL DL** – supports those users who want the maximum expressiveness while retaining computational completeness and decidability.

• **OWL Full** – is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

OWL extends upon RDFS by introducing new language primitives. We do not provide those enumerations as it is not in the scope of this thesis and refer the readers to [Smith et al., 2004]. However, we define one primitive here as it is used throughout the thesis.

OWL allows for stating that two URIs or resources identify the same thing using `owl:sameAs`:

\[
\text{(example:Aftab, owl:sameAs, example:AftabIqbal)}
\]

Here, the statement states that both URIs (example:Aftab and example:AftabIqbal) refer to the same person. Moreover, the `owl:sameAs` statement allows the reuse of external URI in order to create links (conforms to the fourth principle of Linked Data) between different sources.

### 2.1.4 SPARQL

SPARQL is the query language of the Semantic Web. It is “a set of specifications that provide languages and protocols to query and manipulate RDF content on the Web or in an RDF store” [Harris and Seaborne, 2013]. According to Berners-Lee³, “Semantic Web without SPARQL is like relational database without SQL” and therefore, it is considered to be the SQL of the Semantic Web. The SPARQL can be used to query diverse RDF data sources and enables to retrieve and manipulate values from structured and semi-structured data sources. SPARQL 1.0 became the W3C Recommendation in 2008 [Prud’hommeaux and Seaborne, 2008] and SPARQL 1.1 became the W3C Recommendation in 2013.

A SPARQL query comprises of⁴:

• **Prefix Declaration** – for abbreviating URIs.

• **Query Type** – there exist four query types in SPARQL, namely, SELECT to extract raw data, ASK to return boolean values (“yes/no”), CONSTRUCT to return a new RDF graph from query results and DESCRIBE to obtain an RDF graph that describes the queried resource.

• **Query Pattern** – to specify what to query for in the underlying RDF dataset. WHERE clause is used to specify query patterns in SPARQL.

• **Query Modifiers** – to order/rearrange query results.

As an example, the following simple SPARQL SELECT query (see Listing 2.2) when executed on a specific RDF dataset will return “all people who were born in Ireland, ordered by name”.

³ [http://www.w3.org/2007/12/sparql-pressrelease](http://www.w3.org/2007/12/sparql-pressrelease)

⁴ [http://www.cambridgesemantics.com/semantic-university/sparql-by-example](http://www.cambridgesemantics.com/semantic-university/sparql-by-example)
In Listing 2.2, the variables are indicated by a “?” and binding for ?name will be returned by ordered by name using ORDER BY clause. Furthermore, the results of the SPARQL queries can be returned in various formats (XML, JSON, CSV/TSV, RDF and HTML).

### 2.2 RDF VOCABULARIES

In this section, we briefly explain different RDF vocabularies/ontologies that are used in this thesis to represent various software artifacts in RDF.

#### 2.2.1 SKOS

SKOS stands for “Simple Knowledge Organization System” and provides a data model to share and link knowledge organisation systems. It is used to express the structure and content of concept schemes such as taxonomies, folksonomies, classification, thesauri etc. SKOS is an application of RDF that can be used to express a concept scheme, published on the Web and linked with other concept schemes as well as data.

We use SKOS vocabulary in Chapter 5, in order to model keywords/terminologies that are supported by code forges and are used often by software developers to associate metadata to software projects. In Table 2.1, we provide a brief description of a small set of terms taken from SKOS vocabulary that are used in this thesis and leave out the rest where we refer the interested readers to [Miles and Bechhofer, 2009].

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skos:Concept</td>
<td>It is the fundamental element of SKOS vocabulary. A concept can be an idea, object, event etc.</td>
</tr>
<tr>
<td>skos:ConceptScheme</td>
<td>It allows to aggregate one or more SKOS concepts.</td>
</tr>
<tr>
<td>skos:hasTopConcept</td>
<td>It allows to link a concept scheme to the SKOS concepts, contained in a taxonomy or thesauri.</td>
</tr>
<tr>
<td>skos:inScheme</td>
<td>It is used to link the concept schemes with their underlying concepts. It can also be used to link the same concept to several concept schemes.</td>
</tr>
<tr>
<td>skos:prefLabel</td>
<td>It allows to assign a preferred textual label to a concept.</td>
</tr>
</tbody>
</table>

Table 2.1: Description of SKOS vocabulary terms used in the thesis.
2.2.2 FOAF

FOAF stands for “Friend of a Friend” and is a community driven effort to define an RDF vocabulary for describing people, group and documents as well as their relations to each other. FOAF vocabulary is created with a goal to publish a machine-readable Web of data in the domain of personal home pages and social networking. FOAF integrates three kinds of network: social networks, representational networks and information networks.

We use FOAF vocabulary in this thesis to represent software developer information in RDF. As software developer’s information exists in various software development related data sources – such as, in-house software repositories (e.g., bug repository, mailing lists etc.), code forges (e.g., GrrHub, SourceForge etc.) and social media channels (Microblogging, social networking sites etc.) – therefore, we have used FOAF vocabulary in various chapters (Chapter 3, Chapter 4, Chapter 5 and Chapter 6) of this thesis to describe software developers in RDF. In Table 2.2, we provide a brief description of a small set of terms taken from FOAF vocabulary that are used in this thesis and leave out the rest where we refer the interested readers to [Brickley and Miller, 2014].

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>foaf:Person</td>
<td>As it is clear from the term, it is used to represent a person that has several properties (e.g., first name, last name etc.).</td>
</tr>
<tr>
<td>foaf:accountName</td>
<td>It refers to the textual representation of a name (identifier, unique ID) associated with an online account.</td>
</tr>
<tr>
<td>foaf:name</td>
<td>It refers to the name of the resource in textual form.</td>
</tr>
<tr>
<td>foaf:mbox</td>
<td>It refers to the personal mailbox of a person. It is typically identified using the mailto: URI scheme.</td>
</tr>
<tr>
<td>foaf:homepage</td>
<td>It refers to a public Web document, which is not necessarily have to be in HTML format. Usually, the value corresponds to this term refers to the information about its owner.</td>
</tr>
<tr>
<td>foaf:mbox_sha1sum</td>
<td>It refers to the textual representation of the result after applying SHA-1 cryptographic hash function to a mailbox (i.e., mailto:identifier).</td>
</tr>
<tr>
<td>foaf:holdsAccount</td>
<td>It refers a person, group or an organisation that holds an online account.</td>
</tr>
<tr>
<td>foaf:based_near</td>
<td>It refers to a location that is near to something. Moreover, it relates one spatial thing to another, which can be described using geo:lat/geo:long. It can also be used to describe the latitude and longitude of a resource without giving a precise location.</td>
</tr>
<tr>
<td>foaf:knows</td>
<td>It relates a person to another person that he/she knows.</td>
</tr>
<tr>
<td>foaf:primaryTopic</td>
<td>It relates a document to the main thing that the document is about.</td>
</tr>
<tr>
<td>foaf:Document</td>
<td>It relates to anything which is considered to be document.</td>
</tr>
</tbody>
</table>

Table 2.2: Description of FOAF vocabulary terms used in the thesis.

2.2.3 SIOC

SIOC stands for “Semantically Interlinked Online Communities” and is designed to represent the activities of online communities (blogs, forums, mailing lists, wikis, etc.) on the Web. The vocabulary allows to represent the rich data contained in the social web in RDF. It is commonly used with

---

5 description of terms are taken from: [http://xmlns.com/foaf/spec/](http://xmlns.com/foaf/spec/)
2.2 RDF Vocabularies

FOAF vocabulary for describing personal profiles and social networking information. Addition of multiple sub-classes or concepts for different types of Forums or Posts was starting to become unmanageable during the development of SIOC vocabulary and therefore added to the “Types” module of SIOC vocabulary, which is known as “SIOC Types”\(^6\).

We use SIOC vocabulary in this thesis to represent emails and microblog posts (i.e., tweets) in RDF. Therefore, we have used SIOC vocabulary in various chapters (Chapter 3, Chapter 4 and Chapter 6) of this thesis. In Table 2.3\(^7\), we provide a brief description of a small set of terms taken from SIOC and SIOC Types (represented as sioct) vocabulary that are used in this thesis and leave out the rest where we refer the interested readers to [Berrueta et al., 2007].

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sioc:reply_of</td>
<td>It relates an email/post which is a reply or response to another email/post.</td>
</tr>
<tr>
<td>sioc:content</td>
<td>It refers to the textual representation of the content of an email/post.</td>
</tr>
<tr>
<td>sioc:id</td>
<td>It refers to the unique identifier of a SIOC concept.</td>
</tr>
<tr>
<td>sioct:MicroblogPost</td>
<td>It is used to describe a post which is specifically made on a microblogging site.</td>
</tr>
<tr>
<td>sioct:MailMessage</td>
<td>It is used to describe an email which is sent to a mailing list.</td>
</tr>
</tbody>
</table>

Table 2.3: Description of SIOC vocabulary terms used in the thesis.

2.2.4 DCMI

DCMI stands for Dublin Core Metadata Initiative and supports innovation in metadata design and best practices across a wide range of business metamodels. Since 2001, DCMI terms have been published as Web documents and in RDF format. The DCMI vocabulary covers a wide range of domains and can be used to describe Web resources (images, web pages, videos etc.) as well as physical resources (books, CDs etc.). The full set of terms can be found on DCMI website\(^8\).

We have used various terms from this vocabulary in different chapters (Chapter 3, Chapter 5 and Chapter 6) of this thesis. In Table 2.4\(^9\), we provide a brief description of a small set of terms taken from DCMI vocabulary that are used in this thesis and leave out the rest where we refer the interested readers to http://dublincore.org/documents/dcmi-terms/.

2.2.5 MOAT

MOAT stands for “Meaning of a Tag” and aims at providing an easy and collaborative way of semantically-annotating content from free-tagging. It provides a way that allow users to define different meanings (i.e., URIs of resources contained in other knowledge bases) related to a tag. In Table 2.5, we provide brief description of a term taken from MOAT vocabulary that is used in Chapter 6 of this thesis and leave out the rest where we refer the interested readers to http://www.w3.org/2001/sw/wiki/MOAT.

---

\(^6\) [http://www.w3.org/Submission/sioc-spec/](http://www.w3.org/Submission/sioc-spec/)

\(^7\) description of terms are taken from: [http://rdfs.org/sioc/spec/](http://rdfs.org/sioc/spec/)


Table 2.4: Description of DCMI vocabulary terms used in the thesis.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dct:date</td>
<td>It refers to a time period and is associated with an event in the lifecycle of the resource.</td>
</tr>
<tr>
<td>dct:created</td>
<td>It refers to a time period when the resource is created.</td>
</tr>
<tr>
<td>dct:subject</td>
<td>It refers to the topic of the resource and is represented using keywords or key phrases.</td>
</tr>
<tr>
<td>dct:description</td>
<td>It refers to the textual representation of the resource.</td>
</tr>
<tr>
<td>dct:language</td>
<td>It refers to the language that is used to represent the resource.</td>
</tr>
<tr>
<td>dct:creator</td>
<td>It refers to the entity that creates a particular resource. The entity can be a person, group or organisation.</td>
</tr>
<tr>
<td>dct:hasPart</td>
<td>It refers to a referenced resource that includes the described resource either physically or logically.</td>
</tr>
<tr>
<td>dct:isFormatOf</td>
<td>It refers to a referenced resource that is substantially the same as the described resource, but in a different format.</td>
</tr>
</tbody>
</table>

Table 2.5: Description of MOAT vocabulary terms used in the thesis.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moat:taggedWith</td>
<td>It refers to a tag that is used to annotate the resource.</td>
</tr>
</tbody>
</table>

2.2.6 BAETLE

Baetle stands for “Bug And Enhancement Tracking Language”. It is a vocabulary that describes the information kept in bug repositories (e.g., JIRA, BUGZILLA etc.). It is used to specify meta information of bug reports in RDF as well as relating bugs to other software artifacts so that these issues can be tracked. BAETLE vocabulary incorporates and re-uses many concepts from other vocabularies, such as SIOC, FOAF, DCT, SKOS etc., rather than redefining them.

We use BAETLE vocabulary in this thesis to represent bug reports and source control commit logs in RDF. We have used various terms from BAETLE vocabulary in different chapters (Chapter 3, Chapter 4 and Chapter 5) of this thesis. In Table 2.6, we provide a brief description of a small set of terms taken from BAETLE vocabulary that are used in this thesis and leave out the rest where we refer the interested readers to the BAETLE homepage\(^\text{10}\).

2.2.7 DOAP

DOAP stands for “Description of a Project” and is a vocabulary used to describe software projects in a similar fashion as FOAF is designed to describe people. DOAP is specifically used to describe the current state of a software project. One of the use cases of describing software project information using DOAP vocabulary is to allow easy importing of software projects into software directories.

We use DOAP vocabulary in this thesis to represent software project information in RDF. We have used various terms from DOAP vocabulary in Chapter 5 of this thesis. In Table 2.7, we provide

\(^{10}\) https://code.google.com/p/baetle/
Table 2.6: Description of BAETLE vocabulary terms used in the thesis.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baetle:fixes</td>
<td>It refers to a bug report, which has been fixed in a particular source control commit log.</td>
</tr>
<tr>
<td>baetle:modified</td>
<td>It refers to the source code file, which is modified and committed to source control repository through specific source control commit log.</td>
</tr>
<tr>
<td>baetle:author</td>
<td>It refers to a person who has created a resource (e.g., bug report, source control commit log).</td>
</tr>
<tr>
<td>baetle:revision</td>
<td>It refers to a unique identifier assigned to a source control commit log.</td>
</tr>
<tr>
<td>baetle:Committing</td>
<td>It tells that the resource is a source control commit log.</td>
</tr>
<tr>
<td>baetle:summary</td>
<td>It refers to the textual representation of the summary of a bug report.</td>
</tr>
<tr>
<td>baetle:Issue</td>
<td>It refers to a bug report in bug tracking repository.</td>
</tr>
<tr>
<td>baetle:assigned_to</td>
<td>It refers to the person (software developer) who is assigned a particular bug report.</td>
</tr>
<tr>
<td>baetle:reporter</td>
<td>It refers to the person (software developer) who has reported a particular bug report.</td>
</tr>
<tr>
<td>baetle:hasState</td>
<td>It refers to the status (e.g., new, open, started, closed etc.) of a bug report.</td>
</tr>
<tr>
<td>baetle:hasPriority</td>
<td>It refers to the priority (e.g., minor, major, critical etc.) of a bug report.</td>
</tr>
<tr>
<td>baetle:added</td>
<td>It refers to the source code file, which is newly added to the source control repository through specific source control commit log.</td>
</tr>
<tr>
<td>baetle:date</td>
<td>It refers to a time period and is associated with an event in the lifecycle of the resource.</td>
</tr>
</tbody>
</table>

a brief description of a small set of terms taken from DOAP vocabulary that are used in this thesis and leave out the rest where we refer the interested readers to the DOAP vocabulary homepage11.

2.3 MINING SOFTWARE REPOSITORIES

The Mining Software Repositories (MSR) field analyzes and integrates the valuable data available in software repositories in order to discover interesting facts about software systems and projects [Diehl et al., 2009]. Considerable research efforts have focused on studying these software repositories in order to understand the social relation amongst software developers [Bird et al., 2006]. Social network analysis has been extensively used to discover communication patterns amongst software developers and to predict software defects [Meneely et al., 2008]. Moreover, research has also focused on increasing awareness amongst co-workers or software developers in teams to track relevant activities [Gutwin and Greenberg, 1999]. Last but not least, evolution of a software project has been studied by integrating data from different software repositories as discussed by [Antonio et al., 2005; Fischer et al., 2003].

In short, there is plenty of research done on integrating software repositories in order to solve complex issues of software development. Although, the list of research contributions in this domain is quite comprehensive but we present only a small fraction of research in the following. [Sarma et al., 2009] presented Tesseract, a socio-technical dependency browser that enables exploration of cross-linked relationships between different software artifacts and developers. Tesseract is de-

11 https://github.com/edumbill/doap/wiki
Table 2.7: Description of DOAP vocabulary terms used in the thesis.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doap:Project</td>
<td>It is used to state that the resource is a software project.</td>
</tr>
<tr>
<td>doap:name</td>
<td>It refers to the name of the software project in textual format.</td>
</tr>
<tr>
<td>doap:created</td>
<td>It refers to a time period when the resource is created.</td>
</tr>
<tr>
<td>doap:description</td>
<td>It refers to the long (2-4 sentences) textual description of a software project.</td>
</tr>
<tr>
<td>doap:developer</td>
<td>It refers to a person who is working on a software project.</td>
</tr>
<tr>
<td>doap:homepage</td>
<td>It refers to the homepage URL of a software project.</td>
</tr>
<tr>
<td>doap:license</td>
<td>It refers to the software license associated with a software project, under which it is distributed.</td>
</tr>
<tr>
<td>doap:programming-language</td>
<td>It refers to the programming language used to implement the software project.</td>
</tr>
<tr>
<td>doap:repository</td>
<td>It refers to the source code repository URL.</td>
</tr>
<tr>
<td>doap:blog</td>
<td>It refers to the URL of a blog where relevant information about a software project can be found.</td>
</tr>
<tr>
<td>doap:shortdesc</td>
<td>It refers to the short (8 or 9 words) plain textual description of a software project.</td>
</tr>
<tr>
<td>doap:download-page</td>
<td>It refers to the Web page URL from which the software project can be downloaded.</td>
</tr>
</tbody>
</table>

signed to explore the social as well as technical relationships between software project entities by analysing different project archives. It enables cross-links among software artefacts and allows exploration of the relationship and how they changed over time.

[de Souza et al., 2007] propose Ariadne, a plug-in for Eclipse. Ariadne aims to identify the social and technical dependencies and facilitates the coordination and execution of software development activities. Ariadne uses authorship information of source code and translates technical dependencies among source code into social dependencies among software developers. By identifying social dependencies among software developers, Ariadne is able to identify software developers who are more likely to collaborate based on similar technical dependencies. Ariadne identifies the technical dependencies among source code by constructing call-graphs. Ariadne leverages Eclipse’s SearchEngine API to generate call-graph and describes dependencies among source code, which uncovers social dependencies among software developers.

[Cubranic et al., 2005] propose a tool called Hipikat, which indexes data from different software repositories (source code, bug reports, emails) to form a project memory. While working on a particular task, software developer can query Hipikat to display relevant information. For example, a software developer interested in a bug can query Hipikat about other relevant information that exists in the project memory. Hipikat is intended to assist newcomers in the software project by recommending artefacts from the project memory that are relevant to their particular task. Tools like Hipikat makes it easy and very effective for newcomers to “learn from past” when expert software developers are not available.

2.3.1 Semantic Web Based Software Data Integration

Researchers have also taken into account Semantic Web technologies to integrate data from different software repositories. Semantic technologies allow us to transform software repositories into
a conceptually organized and interlinked data space [Heath and Bizer, 2011], incorporating data from different software repositories [Damljanovic and Bontcheva, 2008]. For example, a related work concerning the combination of software repositories has been proposed by [Damljanovic and Bontcheva, 2008]. The data is extracted from different repositories using a crawler and annotated based on Key Concept Identification Tool (KCIT\textsuperscript{12}). KCIT retrieves key concepts from software related legacy content with respect to a domain ontology. To interlink documents based on mentions of key concepts, the authors have used the PROTON KM ontology\textsuperscript{13}. [Ankolekar et al., 2006] propose Dhruv, which is a Semantic Web system to support problem-solving processes in Web communities. The main focus of their research is how open source communities deals with bugs in their software under development. Dhruv helps to cross-links the communication among developers (using forums and mailing lists) with bug reports and source code of the software. They developed an ontology to model software, developers and bugs. The ontologies are semi-automatically populated with data from different information sources to support bug resolution. Their approach assists the communication between software developers for bug resolution and is specifically designed for bug resolution processes.

[Antunes et al., 2007] have presented SRS, a Semantic Reuse System, designed to store and reuse knowledge for software development. It aims to provide efficient ways to store, search and manage the software knowledge. The knowledge about different software elements is named as SDKE (Software Development Knowledge Element). The different elements are represented using Representation Ontology, mapped with the concepts of Domain Ontology and stored in the SDKE, managed using Apache Lucene. Concepts are extracted from software repositories using linguistics tools from Natural Language Processing (NLP) [Jackson and Moulinier, 2002] prior to indexed by Apache Lucene. [Kiefer et al., 2007] have presented EvoOnt\textsuperscript{14}, a software repository data exchange format based on OWL [Smith et al., 2004]. Ontologies are designed to integrate data from bug repositories, electronic communications (i.e., mailing list, discussion forums etc.), source control repositories and source code into a unified semantic model. The authors proposed and used the iSPARQL\textsuperscript{15} engine, which is an extension of SPARQL, to query for similar software entities. Moreover, iSPARQL is based on virtual triples that are used to configure similarity joins [Cohen, 2000].

2.4 SOCIAL ASPECTS OF SOFTWARE DEVELOPMENT

As mentioned in [Iqbal et al., 2009b], software developers use a variety of software tools with underlying repositories to interact with each other or to solve software related problems:

1. **Mailing lists** are used primarily by software developers to communicate and coordinate with other software developers;

2. **Bug repositories** are used by software developers to discuss bug related discussions with others, by posting comments;

3. **IRC Channels** are also used by software developers for quick discussion around a particular software development problem.

\textsuperscript{12} http://www.tao-project.eu/researchanddevelopment/demosanddownloads/content-augmentation-software.html#kcit
\textsuperscript{13} http://proton.semanticweb.org/2005/04/protonkm
\textsuperscript{14} http://www.ifi.uzh.ch/ddis/evo/
\textsuperscript{15} http://www.ifi.uzh.ch/ddis/isparql.html
These repositories have been analyzed in depth to identify communication and coordination issues among software developers by analyzing the social network structure of software developers. The social structure of software developers are usually investigated by examining the patterns of communication among them over the period of time. One can easily understand the development team practices by examining their social structure. Moreover, it provide insights into the coordination, control and socialization aspects.

Social structures of software developers are usually assessed around two aspects: individuals and their interactions. Individuals can be taken into account to understand an overall picture of the group: is it small, large, shrinking or growing? The second aspect is to identify the patterns of interactions between individuals: who is communicating with whom and how often? Patterns in communication among software developers are usually used to look for answers to questions of coordination, control and socialization. Figure 2.2 shows interactions among software developers of squirrelmail software project.

![Pattern of interactions among software developers of squirrelmail project](http://squirrelmail.org/)

In Figure 2.2, the nodes represent software developers and the lines between nodes represents their interactions (i.e., social relations) with each other. Moreover, such a network of interactions can be further investigated in order to identify the social behavior of software developers with others.

---

16 http://squirrelmail.org/
2.4 Socio-Technical Dependencies among Software Developers

Research studies in the past have identified that technical dependencies among source code creates social dependencies between software developers [de Souza et al., 2004b; Grinter, 2003]. For example, given two different software modules that interact with each other, the software developers responsible for developing each software module are socially dependent and are required to interact regularly to make sure that their work is aligned. Based on this observation, [de Souza et al., 2004a] argue that the source code itself can be used to identify the need of social interactions that must happen among software developers in order to guarantee a smooth flow of work.

2.4.1 Technical Dependencies

Dependencies among pieces of source code exist because they make use of functionality provided by others. For example, a software module $A$ uses the functionality of software module $B$, as a result $A$ depends on $B$. This means that software module $A$ depends on the functionality of software module $B$ in order to perform its functionality. Dependency relations of a software project can be identified using call-graph, because it contains information about which software module calls which other software module. Figure 2.3 shows example of a small Java program. A directed edge from one node to another node indicates the calling behavior of a method by another method. For example, a directed edge between $B$ and $D$ indicates that source code $B$ depends on source code $D$ because it uses the functionality of source code $D$.

Figure 2.3: An exemplary call graph of a Java program.

2.4.1.2 Socio Dependencies

By extracting technical dependencies among source code (cf. Section 2.4.1.1), the graph unveils social dependencies among software developers responsible for developing those source code. Therefore, authorship information of each source code can be extracted from source control repository as it stores information about the changes applied, date and time of these changes, software developer who applied those changes, etc. A social call-graph [de Souza et al., 2004a] can be derived by combining information from a call-graph with authorship information present in the source control repository. This would reveal social dependencies among software developers for a given
piece of source code. For instance, take the example of a social call-graph in Figure 2.4, where directed edges between software developers and source code depicts authorship information. The figure shows that source code $B$ depends on source code $D$ and dev-4 is the developer of source code $B$ while dev-7 is the developer of source code $D$. Therefore, we can say that dev-4 depends on dev-7. Hence, these software developers are required to communicate and coordinate in order to align their work.

![Figure 2.4: An exemplary socio dependencies graph showing information about software developers who are developing interdependent source code.](image)

Social network graph describing the dependency relations among software developers can be derived from social call-graph, which describes technical dependencies among source code as well as authorship information. Such a graph shows direct edges only between those software developers who are dependent on each other due to the technical dependencies among source code they are developing. In Figure 2.5, a direct edge between dev-4 and dev-7 indicates that both software developers are socially dependent due to the technical dependency between source code $B$ and $D$. Further, we see in Figure 2.4 that dev-5 and dev-8 are working on the same source code file (i.e., $C$), therefore they are also socially dependent on each other.

![Figure 2.5: An exemplary social network graph showing dependencies among software developers.](image)
“A key issue in current computer-aided software engineering (CASE) environments is the desire to link tools that address different aspects of the development process.”

— Anthony I. Wasserman [Wasserman, 1990]

During the software development process, a variety of software artifacts are produced as shown in Figure 3.1. These software artifacts are contained within different software repositories that are part of a particular software project. Some of these software repositories are directly under the control of software developers, whilst others are shared among the users and software developers of a software project. Examples of these software repositories are:

1. **source control repositories** store latest changes made to the source code, documentation etc., relevant to a particular software project. It describes the change history of the software artifacts from the initial check-in until the current release. It also keeps information about the software developers who have made changes to the source code as development progresses as well as information about bugs, which have been fixed in certain commits. The information about bugs are usually provided by software developers themselves by inserting the bug IDs in the description while making a commit to the source control repository.

2. **bug repositories** keep track of the software defects of a particular software project as well as the software developers who have reported and assigned those defects. It also stores information about the source control commits, which fixes a particular bug as well as source code snippets posted by software developers while commenting on a particular bug.

3. **archived communications** between software developers record rationale for decisions made by software developers throughout the life-cycle of a software project. Often, bugs and source control commits are discussed by software developers through exchanging emails.

4. **wikis** are used by software developers to maintain documentation and any other knowledge relevant to a software project at one place.

In Table 3.1, we summarize the key software artifacts that are found in different software repositories and are related to a particular software project. We break down the software repositories into different sources of information and for each source, we show the different software artifacts it contains.

Table 3.1 shows that same type of software project related information (i.e., software artifacts) is available through different software repositories. Mining the information stored in these software repositories, practitioners can depend less on their intuition and experience, and more on historical and field data [Hassan, 2008]. There are plenty of use case scenarios that requires the integration of information contained inside these software repositories, some of which are listed below:

**Identifying change couplings and social dependencies among software developers**
Figure 3.1: Different software artifacts involved in the software development process, including its participants. We do not consider software artifacts that are highlighted in grey color while we present our integration approach later in this chapter. However, our approach can be extended and applied to all software artifacts involved in the software development process.

Technical dependencies in the source code are usually identified by analyzing how often they are committed together on a source control repository. Source code files that are created or edited, for example, while fixing a bug or adding a new feature are often checked-in together as one change set [Weber, 1997]. By discovering the technical dependencies amongst source code files, we can discover the socio-technical dependencies amongst software developers by retrieving authorship information of those source code files [de Souza et al., 2007].

Lack of awareness among co-workers

Software developers are often keen to keep track of other team mates’ work activities. While working on a particular piece of code base, software developers often felt disconnected from the rest of the community or lose track of co-worker’s activities. Therefore, software developers often use the project mailing lists to maintain awareness of what is being discussed among software developers and also use it to coordinate their work. Software developers also obtain awareness through meetings to keep aware of problems co-workers are working on. The study carried out by [Ko et al., 2007] reveals that the most frequently acquired information by a software developer includes: what a co-developer has been working on?

Information relevant to a particular task

While working on a particular piece of source code, a software developer is often interested in other information relevant to the task in hand. For example, a software developer is in-
Table 3.1: Different types of software artifacts found in each software repository. It shows that information about software artifacts is distributed across different software repositories and there exists an implicit connection between the information contained inside these software repositories.

<table>
<thead>
<tr>
<th>Software Repository</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>source code</td>
<td>software developers, classes, methods, software packages</td>
</tr>
<tr>
<td>source control</td>
<td>software developers, source code, documentation, bugs, commits</td>
</tr>
<tr>
<td>bug tracking</td>
<td>software developers, source code, bugs, commits</td>
</tr>
<tr>
<td>mailing list</td>
<td>software developers, source code, bugs, commits</td>
</tr>
<tr>
<td>documentation</td>
<td>software developers, source code</td>
</tr>
</tbody>
</table>

Table 3.1: Different types of software artifacts found in each software repository. It shows that information about software artifacts is distributed across different software repositories and there exists an implicit connection between the information contained inside these software repositories.

The software repositories listed in Table 3.1 shows that software development usually takes place in two environments, (i) the Integrated Development Environment (IDE), such as Eclipse\(^1\), and (ii) the Web, for finding source code examples and documentation, logging or updating issues on bug repositories, participating in discussions on forums or mailing lists etc.; as a large. We need hence not only make the links between the software repositories within a software project explicit but also allow connecting to other relevant data on the Web. For example, explicitly linking the discussions relevant to a bug report that are exchanged among software developers on the project mailing list and further linking it to the source control commit log, which fixes that particular bug. Having such an explicit representation of the connection between the software artifacts available, we would be able to support certain use case scenarios often found in the software development process:

**Synthesis Scenarios**—support the development of new source code:
- A software developer will effectively query colleagues for support (expert finding) and/or being suggested contextualized source code fragment(s);
- A project manager will learn from previous projects and/or metrics.

**Analysis Scenarios**—support the project management and maintenance of existing source code:
- Perform opinion mining on SIOC [Bojars et al., 2008] (Semantically Interlinked Online Communities) based representations of the mailing lists or discussion forums [Softic

\(^1\) [http://www.eclipse.org/]
and Hausenblas, 2008] in order to generate reports on a component or extract feature requests and bug reports.

- Given that the documentation is interlinked with the source code, a dynamic FAQs can be provided;
- Software developer profiles, based on their commits to the source control repository and the source code can be provided.

Each software repository (cf. Table 3.1) serves a specific need and provides a unique view on the project. As mentioned by [Ramler and Wolfs, 2008], these individual views have to be merged in order to get a comprehensive overview. However, there are obstacles underway for seamless software repositories integration as discussed in Section 1.3.1. Therefore, there is a need to apply a standard approach, which makes it easier to integrate information contained inside distributed and heterogeneous software repositories. Hence, we require methods and techniques to integrate relevant software artifacts across different software repositories.

Linking software artifacts distributed across software repositories require a standard mechanism to specify the existence and meaning of connection between them. This mechanism is well supported by Resource Description Framework (RDF) [Klyne et al., 2004]. RDF provides a flexible way to describe entities and how are they related to each other. For example, to publish information about source control commit logs relevant to a software project in RDF, we first need to define distinct URI for each source control commit log. Later, we publish information about each source control commit log as a set of statements stating its revision number, who made the commit, what sources were updated/added and the time-stamp value. Moreover, if a source control commit log fixes a particular bug and we insert a statement stating that it fixes a bug (represented by a URI), then we are creating a Web of interlinked information. This gives us a number of benefits:

- the software artifacts are placed in context having a unique URI through which it can be annotated and referenced;
- the interlinked software artifacts enable information to be combined across distributed software repositories.

In the rest of this chapter, we discuss the publication and integration of software repositories (based on software artifacts) as Linked Data and further show how the interlinked information can be delivered to software developers in their preferred development environment. Therefore, the remainder of this chapter is structured as follows:

- In Section 3.1, we discuss basic concepts of RDF and provide an example of representing software artifact in RDF using existing RDF vocabularies. Additionally, we provide an interlinking example connecting various software artifacts conceptually in order to show the value of integrating software artifacts.
- In Section 3.2, we introduce Linked Data Driven Software Development (LD2SD), a lightweight Semantic Web methodology to turn software artifacts – such as data from source control repositories, bug repositories or Java source code – into Linked Data. We explain in detail our methodology to publish different software repositories as RDF datasets as well as integrating those software repositories by interlinking their underlying RDF datasets. We further show the benefit of our interlinking methodology by publishing and interlinking RDF datasets from different software repositories of 5 Apache projects.
• Next, in Section 3.3, we present how the interlinked RDF datasets can be exploited by software developers using a plugin for an Integrated Development Environment (IDE), such as Eclipse. We further evaluate the effectiveness of our automated approach against traditional ways (manual approach) of finding relevant information from different software repositories by carrying out a user study.

3.1 INTEGRATING SOFTWARE ARTIFACTS: LINKED DATA APPROACH

In order to provide a uniform access to the different software repositories, one needs to integrate software artifacts contained inside those software repositories. As motivated and discussed in Section 1.4, we propose a Linked Data driven approach towards achieving this goal. The two main technologies that support Linked Data are HTTP URIs (to identify and retrieve descriptions of entities) and RDF (to describe entities using a graph-based data model and define relationships between them). RDF is originally designed to describe meta data and to model information about any resource. The RDF graph model provides a flexible way to represent information of any real-world domain in the form of statements and how are they related to each other. However, RDF data publishers are required to define their own vocabularies, which they plan to use while describing knowledge/information in the form of RDF statements. This is required specifically for RDF data publishers to indicate which types (i.e., classes) of resources and their properties are used while describing knowledge [Manola et al., 2004].

RDF statements can be specified by means of triples and are expressed as tuples of the form:

\[(subject, predicate, object)\]

which, can be used to specify type (i.e., class) to a resource:

\[(123, type, Bug)\]

to define attribute of a resource:

\[(123, title, "null pointer exception")\]

and/or to define relationship between resources:

\[(123, fixedBy, Tom)\]

In the context of describing software artifacts as RDF, data publisher can describe a class as Bug (to represent a bug report), and use properties such as hasPriority, reporter, summary to describe further information about a bug report. Similarly, data publisher can describe other software artifacts through classes such as Commit (to represent a source control commit) and MailMessage (to represent an email), and properties such as modified, revision, Date, sender to describe further information about these software artifacts.

RDF does not provide any means to describe domain-specific classes and properties. However, it can be described as an RDF vocabulary, using extensions that are provided to RDF, referred to here as RDF Schema [Brickley and Guha, 2004]. RDF Schema does not provide a vocabulary to define domain-specific classes and properties but provides a mechanism to define them and to indicate
which classes and properties can be used together (for example, to say that the property sender will be used while describing a MailMessage). For further details on how to use RDF Schema in order to define your own RDF vocabularies, we refer the readers to [Brickley and Guha, 2004; Muñoz et al., 2009].

Naming is an important part of RDF. User-defined resources are thus optionally named using a URI, however, unnamed resources are usually represented as blank-nodes. Therefore, the character string 123 can be named using a URI in order to make it an ideal Web-scope identifier. Most RDF syntaxes allow to use Compact URI (CURIE) names of the form prefix:reference to denote URIs. For example, given a prefix ex: which provides a shortcut for the URI (http://example.com/ns#), then ex:Bug denotes http://example.com/ns#Bug.

The most prominent RDF term is rdf:type, used for stating that a resource is an instance of a given class. For example, the following statement states that a resource (represented by bug:123) is a bug report:

\[
\text{(bug:123, rdf:type, ex:Bug)}
\]

We have discussed briefly basic notions of RDF and how it can be used to define resources and describe their properties. In the following, we provide an overview of different RDF vocabularies that can be used together in order to represent the information about software artifacts as RDF statements. The RDF community has already defined some RDF vocabularies that covers different aspects of the software development process. Therefore, we use existing RDF vocabularies rather than creating our own RDF vocabularies in order to represent software artifacts through RDF statements. We present 4 different software artifacts, their corresponding RDF vocabularies and some exemplary RDF vocabulary properties that are used to represent those software artifacts as RDF statements in Table 3.2.

<table>
<thead>
<tr>
<th>Artifacts</th>
<th>RDF Vocabularies</th>
<th>Example Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>ld2sd(^a), dct(^b)</td>
<td>ld2sd:JavaClass, ld2sd:package, ld2sd:hasMethod, dct:title</td>
</tr>
<tr>
<td>bug</td>
<td>baetle(^c), foaf(^d), dct</td>
<td>baetle:Issue, baetle:reporter, baetle:assigned_to, foaf:name, dct:created</td>
</tr>
<tr>
<td>commit</td>
<td>baetle, dct</td>
<td>baetle:Committing, baetle:author, baetle:modified, baetle:revision, dct:date</td>
</tr>
<tr>
<td>email</td>
<td>email(^e), sioc(^f), sioc:t, foaf, dct</td>
<td>sioc:MailMessage, sioc:reply_of, email:from, dct:date, foaf:mbox</td>
</tr>
</tbody>
</table>

Table 3.2: RDF vocabularies and their exemplary properties that are used to represent software artifacts as RDF statements.

\(^{a}\) http://vocab.deri.ie/ld2sd/
\(^{b}\) http://dublincore.org/documents/dcmi-terms/
\(^{c}\) http://code.google.com/p/baetle/
\(^{d}\) http://xmlns.com/foaf/spec/
\(^{e}\) http://simile.mit.edu/2005/06/ontologies/email/
\(^{f}\) http://sioc-project.org/ontology
\(^{g}\) http://rdfs.org/sioc/types

In the following, we consider bug as a software artifact and provide an example on how the corresponding RDF vocabularies are used to represent information about a bug report as RDF
3.1 Integrating Software Artifacts: Linked Data Approach

Figure 3.2: Graphical representation of a bug report in RDF by using classes and properties from baetle, foaf and dct RDF vocabularies.

statements. baetle (Bug And Enhancement Tracking Language) vocabulary is designed by the community to describe the information kept in bug tracking repositories such as Bugzilla, JIRA etc. Often, concepts/classes are re-used from existing RDF vocabularies while creating a new RDF vocabulary. For example, to represent a bug report in RDF, baetle vocabulary re-uses concepts from foaf (Friend of a Friend), sioc (Semantically Interlinked Online Communities) and dct (Dublin Core Metadata Initiative) vocabularies. The reason of choosing other vocabularies is to re-use classes and properties defined by them. A bug report usually contains the following information: description about the bug, priority of the bug, person who reported the bug, date when the bug report was created etc. Therefore, to represent person’s information who reported the bug, baetle vocabulary uses foaf vocabulary that allows to describe people, however, bug related information is described using baetle vocabulary as shown in Figure 3.2. An exemplary representation of a bug report as well as other software artifacts in RDF format using different RDF vocabularies (outlined in Table 3.2) are discussed in detail in Section 3.2.3.

Once the software artifacts contained inside different software repositories are modeled using RDF vocabularies and published as RDF datasets, the next step is to bridge the connection between RDF datasets by inserting relations among them in the form of RDF triples. Having lack of inter-connection between RDF datasets prevent the navigation to or discovery of relevant information as human users or software agents are able to do in the traditional Web by following links. Therefore, we are required to identify links/relations between RDF datasets. In the following, we provide an example that describes the relationship between a bug report and a source control commit log and further show how both software artifacts can be connected to each other.
Bugs are often fixed by modifying source code and later, changes are pushed to the source control repository. Software developers often mention bug IDs in the summary of a source control commit log while committing changes to the source control repository, if the change fixes a particular bug. This implies a relation between source control commit log and bug report. This implicit connection can be made explicit by inserting an RDF statement. For example:

```
svn:456 baetle:fixes bug:123;
```

where `baetle:fixes` is a link (a.k.a RDF link) between source control commit log and bug report. It states that a source control commit log (456) fixes a bug report (123). Unlike HTML links which indicate that two documents are somehow related and leave mostly up to the reader to infer the nature of relationship, RDF link enables the data publisher to explicitly specify the nature of relationship between two entities. We provide another example, which defines a link between an email and a bug report if the email mentions a bug ID. For example:

```
email:345 rdfs:seeAlso bug:123;
```

states that an email with ID (345) mentions a bug report (123). In Figure 3.3, we present a high level overview of potential links between various software artifacts that are contained inside different software repositories based on their implicit relationship with each other.

![Diagram](image.png)

Figure 3.3: An example depicting connections among different software artifacts. Connections between software artifacts are made explicit by inserting RDF statements, which allows the user to navigate across various software artifacts.
Given that the software artifacts are interlinked across different software repositories (cf. Figure 3.3), we are able to execute a single query in order to retrieve information from different software repositories. For example, based on the links between software artifacts depicted in Figure 3.3, we are able to retrieve not only meta information about a particular bug report but can navigate to the source control commit log which tells us the source code file that was modified in order to fix that particular bug report as well as an email where this particular bug report was originally communicated by a software developer. More specifically, the interconnected software artifacts are easily exploitable to answer certain type of questions:

- which source code files were committed while fixing bug report #123?
- are there any discussions related to a bug report #123 exchanged among software developers on the mailing list?
- who fixed the bug report #123?

Taking into account the first question listed above, we are able to exploit the owl:sameAs link between source control commit log and bug report in order to find out the commit log where this particular bug report was fixed by a software developer. Once the commit log is identified, we are able to easily pull information about the source code files which were modified in order to fix that particular bug report.

In this section, we have provided basic concepts of RDF to represent real-world domain information as RDF statements. Further, we outlined different RDF vocabularies and showed an example of representing software artifact using these RDF vocabularies. Additionally, we provided an example connecting different software artifacts conceptually in order to demonstrate the added value of it. In the next section, we take into consideration some software artifacts and provide details on how to convert them into RDF datasets. Once the software artifacts are converted into RDF datasets, we expand on our interlinking approach with examples to show integration across different software repositories by establishing links between different software artifacts as shown in Figure 3.3.

3.2 LINKED DATA DRIVEN SOFTWARE DEVELOPMENT (LD2SD)

“to the best of my knowledge, combining Linked Data and software engineering artifacts is an original idea that has been explored in first place by the authors”

— Reviewer of the LD2SD paper, 2009

As mentioned earlier, RDF provides a flexible way to describe software artifacts and how are they related to each other. Therefore, we decide to realize a Linked Data driven approach—known as Linked Data Driven Software Development (LD2SD)—for integrating software repositories, as we have argued in [Iqbal et al., 2009b]. We provide a Linked Data driven methodology to convert software artifacts contained inside different software repositories into RDF datasets and interlink them where necessary. We show examples of various software artifacts that are converted to RDF datasets and later, how these software artifacts can be interlinked to each other. We apply our methodology to some selected Apache projects in order to demonstrate the rich amount of implicit connections that exists among different software repositories (based on their underlying software artifacts) and argue that it can be made explicit through applying a Linked Data approach.
3.2.1 LD2SD Foundations

We first introduce Linked Data, the foundation of Linked Data Driven Software Development (LD2SD), and then give an overview on the LD2SD methodology.

3.2.1.1 Linked Data

Linked Data is about applying the principles of the Web to share data and practicing that at a deeper level than just putting up everything in one big database or file. The basic idea of Linked Data [Bizer et al., 2008] has first been outlined by Tim Berners-Lee in 2006 [Berners-Lee, 2006], where he described the Linked Data principles as follows: (i) all items should be identified using URLs [Sauermann and Cyganiak, 2008] and these URLs should be dereferenceable, that is, (ii) using HTTP URLs allows looking up an item identified through the URI, further (iii) when looking up an URI (an RDF property is interpreted as a hyperlink), it leads to more data, and (iv) links to URIs in other datasets should be included in order to enable the discovery of more data. In contrast to the full-fledged Semantic Web vision, Linked Data is mainly about publishing structured data in RDF using HTTP URIs rather than focusing on the ontological level or inferencing. This simplification—comparable to what the World Wide Web did for hypertext—fosters a wide-spread adoption [Ayers, 2007].

3.2.1.2 Methodology

With “Linked Data Driven Software Development” (LD2SD), we propose a Linked Data based, light-weight framework that allows to integrate software artifacts found in different software repositories. The so created interlinked data space enables uniform query and browsing facilities. The overall concept of LD2SD is to extract information from software repositories, describe that information in RDF and interlink them where necessary, yielding interlinked datasets. Later, the integrated information is provided to the software stakeholders by developing tools/services for their preferred development environments (e.g., Eclipse). We summarize the steps of our proposed methodology in the following:

1. Assign URIs to software artifacts (i.e., software developers, bugs, source code, commit logs etc.) found in different software repositories and represent them in RDF using Linked Data principles, yielding interlinked datasets. Use HTTP URIs so that useful information about a particular software artifact can be provided upon request. As mentioned earlier and further shown in Table 3.1, a software repository usually contains information about various software artifacts, therefore, use different URI patterns for different software artifacts that are contained in a software repository (see Section 3.2.3 for details) in order to uniquely identify them;

2. Once the software artifacts are published as Linked Data, one can use an indexing engine to index, align and filter interlinked datasets. The main reason of proposing an indexing facility is to provide a single interface to software stakeholders for searching the software artifacts that are hidden inside different software repositories. For indexing purposes, one can use either state-of-the-art and easy-to-use semantic index tools/services such as Sindice [Oren et al., 2008], SWSE [Hogan et al., 2011] or general purpose indexing engine like Apache Lucene/Solr2;
3. Deliver the integrated information to software stakeholders in their preferred development environments by developing tools/plugins, as we discuss later in Section 3.3.

We now provide a reference architecture and implementation of LD2SD based on the above described LD2SD methodology in the following subsection.

3.2.1.3 Architecture

The reference architecture of Linked Data Driven Software Development (LD2SD)—depicted in Figure 3.4—is a layered, Linked Data based integration of software repositories in order to enable software stakeholders to consume the integrated information via different tools/plugins.

Figure 3.4: A reference architecture of LD2SD methodology showing the 3 layers of data publishing & interlinking, indexing and exploitation of integrated information via end-user applications.

In the following, we briefly describe the role of each layer depicted in Figure 3.4:

**Data Layer**

The data layer deals with publishing RDF datasets by extracting data from different software repositories of a software project. Further, the generated RDF datasets are interlinked with each other by inserting owl:sameAs and rdfs:seeAlso type of links. More details on publishing software repositories as RDF datasets is discussed in Section 3.2.3.
INTEGRATION LAYER

After publishing RDF datasets, the next step is to integrate, index and store them. There are different ways to integrate RDF data sources proposed by various researchers [Langegger et al., 2008; Phouc et al., 2009; Quilitz and Leser, 2008; Vdovjak and Houben, 2001]. However, we provide an integration example using DERI pipes [Phouc et al., 2009] in the following.

DERI Pipes is an open source project used to build RDF-based mashups. It allows to fetch RDF documents from different sources (referenced via URIs), merge them and operate on them. In our case at hand, this involves four major steps (cf. Figure 3.5):

1. Fetch the RDF representation of software artifacts (e.g., source code, commit log, bug etc.) using the RDF Fetch operator3;
2. Merge the RDF datasets using Simple Mix operator4;
3. Query the resulting, integrated dataset with SPARQL5;
4. Apply XQuery6 in order to sort and format the results from previous step.

INTERACTION LAYER

The interaction layer handles the interaction between the integrated datasets and the software stakeholders. We discuss on exploiting the interlinked RDF datasets via plug-in for an Integrated Development Environment (IDE) such as Eclipse and Web interface for searching software artifacts in Section 3.3.

3.2.2 LD2SD Characteristics

In 2007, the Linking Open Data (LOD) initiative7, an open, collaborative effort aiming at bootstrapping the Web of Data by publishing datasets in RDF on the Web and creating interlinks between these datasets, had launched. With approximately 295 interlinked datasets offering more than 31 billions of RDF triples and millions of interlinks, the so called “LOD cloud” enables entire new application areas [Hausenblas, 2009a].

We highlight the fact that by utilizing LD2SD, a Web-scale integration of software project related information is hence made possible. One can imagine that—as both LD2SD and LOD follow the Linked Data principles—we are able to connect software artifacts to the LOD datasets, such as DBpedia [Auer et al., 2007], hence enabling the reuse of existing information in the software development process.

3.2.3 Publishing Software Artifacts as RDF Datasets

In order to interlink software artifacts within and across different software repositories, we require a standard mechanism to specify the existence and meaning of connections between different software artifacts contained inside different software repositories. This mechanism is well supported by Resource Description Framework (RDF), which provides a flexible way to describe things in

---

3 http://pipes.deri.org/doc/#FETCH
4 http://pipes.deri.org/doc/#MIX
5 http://www.w3.org/TR/rdf-sparql-query/
6 http://www.w3.org/TR/xquery/
7 http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
3.2 LINKED DATA DRIVEN SOFTWARE DEVELOPMENT (LD2SD) | 51

Figure 3.5: A graphical DERI pipes editor that allows to fetch RDF data sources from user provided URLs, integrate and operate on them.

the world – such as, in the context of software development domain it is software developers, bugs, commits, emails etc., – and how are they related to each other. There are a set of principles which must be followed while representing information in RDF. In the following, we outline and briefly explain these principles in the context of representing information about software artifacts in RDF:

**p1** Identify entities that exists in a software repository;

It is important to identify different entities that exists in a software repository and provides meaningful information. For example, a bug repository contains information about bug reports and software developers who have reported and assigned those bug reports. Therefore, bug report IDs and software developers must be considered as entities.

**p2** Assign globally unique identifiers to these entities;

In a bug repository, each bug report is assigned a different ID in order to distinguish it from other bug reports. Moreover, multiple software developers exists on a bug repository, which reports/fixes those bugs. However, a bug report ID or a software developer name/ID can be similar across different software projects. Therefore, it is necessary to assign globally unique identifiers to these entities in order to differentiate each bug report and software developer across multiple software projects. To publish software artifacts information as RDF on the
Web, HTTP URIs is the best choice as it combines globally unique identification with a simple and well-understood retrieval mechanism [Heath and Bizer, 2011], enabling these URIs to be dereferenced.

**p3** Describe entities and relationships between them using RDF vocabularies;

HTTP URI is dereferenceable, meaning that any one can look up the URI and retrieve a description of the entity that is defined by that particular URI. Therefore, it is necessary to describe each entity as a set of RDF statements and further define its relationship with other entities where applicable. For example, relationship of a software developer to a particular bug report can be either of a assignee or reporter. Therefore, such types of relationships between entities must be represented in RDF using RDF vocabularies.

**p4** Include links to other entities across different RDF datasets.

It is important to connect relevant entities across different RDF datasets in order to glue them into a global, interconnected data space. This enable applications to discover additional information about entities. In the context of Linked Data, entities are connected to each other through RDF links. For example, a source control commit log fixes a particular bug report and we establish an RDF link between that source control commit log and relevant bug report. This enables us to query, who fixed the bug report and which source code files were modified while fixing that bug report.

In the rest of this section, we consider the first three principles and discuss them in detail with examples of representing different software artifacts (i.e., source code, bug, commit log, email etc.) as RDF datasets. We explain the fourth principle with examples in Section 3.2.4.

3.2.3.1 *Publishing Source Code as RDF Dataset*

A source code consists of the programming statements that are created by a software developer using a text editor and saved in a file. The source code of a software project is usually held in many source code files and are carefully arranged into a directory tree, also known as source tree. In object oriented programming, a source code file consists of a class (in some cases multiple classes) which contains a set of methods and variables. A class usually imports external libraries in order to reuse functionality of other software components. In a nutshell, source code of a software project contains information about classes, methods, variables, documentation (i.e., javadocs), developer names etc. Listing 3.1 shows a Java source code snippet taken from the Apache Tomcat project.

In the following, we explain how to publish source code as an RDF dataset based on the principles that are outlined in the beginning of this section.

**p1** *Identify entities that exists in a software repository*

In order to represent meta information about source code in RDF according to the Linked Data principles, we first identify the different entities that exists in source code. Considering the exemplary source code snippet listed in Listing 3.1, we classify software package, class, developer and method as entities in the source code.

---

package org.apache.jasper;

import javax.servlet;

/**
 * Base class for all exceptions generated by the JSP engine. Makes it
 * convenient to catch just this at the top-level.
 *
 * @author Konstantin Kolinko
 */

public class JasperException extends ServletException {

    private static final long serialVersionUID = 1L;

    /**
     * Creates a JasperException with the embedded exception
     *
     * @param exception
     */
    public JasperException (Throwable exception) {
        super (exception);
    }
}

Listing 3.1: Java source code snippet taken from the APACHE TOMCAT project.

p2 Assign globally unique identifiers to these entities

Once entities are identified in the source code, we assign globally unique identifiers to these entities. In particular, we assign HTTP URIs to these entities so that useful information about each entity can be provided upon request. As an example, we present most commonly found entities in Java source code (based on Table 1.2) and their exemplary HTTP URI patterns (in accordance with the 1st and 2nd principle of Linked Data) in Table 3.3. Most programming languages do possess similar kind of information, therefore, similar kind of entities can be easily identified in the source code written in different programming languages.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
<th>Example URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>Apache TOMCAT</td>
<td><a href="http://example.org/tomcat">http://example.org/tomcat</a></td>
</tr>
<tr>
<td>software</td>
<td>org.apache.jasper</td>
<td><a href="http://example.org/tomcat/java/org/apache/jasper">http://example.org/tomcat/java/org/apache/jasper</a></td>
</tr>
<tr>
<td>package</td>
<td>JasperException</td>
<td><a href="http://example.org/tomcat/java/org/apache/jasper/JasperException">http://example.org/tomcat/java/org/apache/jasper/JasperException</a></td>
</tr>
<tr>
<td>developer</td>
<td>Konstantin Kolinko</td>
<td><a href="http://example.org/tomcat/author/Konstantin_Kolinko">http://example.org/tomcat/author/Konstantin_Kolinko</a></td>
</tr>
<tr>
<td>method</td>
<td>JasperException()</td>
<td><a href="http://example.org/tomcat/java/org/apache/jasper/JasperException#JasperException">http://example.org/tomcat/java/org/apache/jasper/JasperException#JasperException</a></td>
</tr>
</tbody>
</table>

Table 3.3: URI patterns for different types of entities found in Java source code.

In order to assign globally unique identifier to a source code, we use project name in the URI (e.g., we have used “tomcat” in the URIs in Table 3.3) in order to avoid conflicting URIs if different software projects have similar software packages or class names. As a software developer often work on multiple software projects, therefore, we also use project name in the URI of a software developer in order to keep track of the activities of a software developer within a particular software project. Note that the domain name (i.e., example.org) used in the URI patterns listed in Table 3.3 is for illustration purposes only and must be replaced by the domain name of a data publisher.

p3 Describe entities and relationships between them using RDF vocabularies
Once the HTTP URIs are assigned to the entities in the source code, the next step is to define the relationships between them. For example, the source code in a Java software project usually comprises of software packages (i.e., software modules) and each software package comprises of different classes. Therefore, we can define the relationships between software packages and their underlying classes. Taking into account the example of a Java source code shown in Listing 3.1, we can say that "JasperException.java" belongs to the software package "org.apache.jasper". Similarly, we can describe the relationship of a particular class to its defined methods and software developers that are working on that particular class. For example, "JasperException()" is a method defined in "JasperException.java" class and "Konstantin Kolinko" is the developer of "JasperException.java" class. These statements of relationships contained inside a source code are, in essence, links between entities. RDF allows us to define these relationships explicitly and publish this information in a form so that other entities can discover and link to it for further use. In order to show how RDF transformation of source code may look like, we provide an example of RDF representation of Java source code based on the source code snippet (cf. Listing 3.1) taken from the Apache Tomcat project in Listing 3.2.

```rml
@prefix : <http://example.org/tomcat/java/> .
@prefix class: <http://example.org/tomcat/java/org/apache/jasper/> .
@prefix method: <http://example.org/tomcat/java/org/apache/jasper/JasperException#> .
@prefix ld2sd: <http://vocab.deri.ie/ld2sd#> .
@prefix dct: <http://purl.org/dc/terms/> .

class:JasperException a ld2sd:JavaClass;
dct:title "org/apache/jasper/JasperException";
ld2sd:comment "Base class for all exceptions generated by the JSP engine. Makes it convenient to catch just this at the top-level.";
ld2sd:author <http://example.org/tomcat/author/Konstantin_Kolinko>;
ld2sd:package <http://example.org/tomcat/java/org/apache/jasper>;
ld2sd:imports "javax.servlet";
ld2sd:hasMethod method:JasperException;
ld2sd:hasAttribute "serialVersionUID"
.
method:JasperException a ld2sd:JavaMethod;
dct:title "org/apache/jasper/JasperException#JasperException"
ld2sd:comment "Creates a JasperException with the embedded exception"
ld2sd:parameter-type "java.lang.Throwable";
ld2sd:parameter-name "exception"
.
```

Listing 3.2: RDF representation of Java source code based on Listing 3.1.

As mentioned earlier, we assigned URIs to entities that exist in a source code and also defined the association between them. In Listing 3.2, the relationship between a class and a software developer is represented as RDF statement using ld2sd:author property, which tells that "Konstantin Kolinko" (represented in the form of URI as: http://example.org/tomcat/author/Konstantin_Kolinko) is a developer of "JasperException.java" (represented in the form of URI as: http://example.org/tomcat/java/org/apache/jasper/JasperException). Similarly, we have defined the relationship of "JasperException.java" to its software package and methods by using ld2sd:package and ld2sd:hasMethod properties respectively. Moreover, we have used several other properties to describe meta information (such as javadocs, import statements etc.) of "JasperException.java", as shown in Listing 3.2.
3.2.3.2 Publishing Source Control Repository as RDF Dataset

Source control repository (a.k.a. versioning system) is designed to track changes within a set of directories or files. In the context of software development, source control repository is used to manage up-to-date version of source code and documentation related to a software project. A source control commit log usually contains information about: who (author) made the commit and when (time-stamp), what was changed (modified, added or deleted) and a brief description that explains the reason behind the changes made. When a commit log is made to the source control repository, a unique revision ID is assigned to that commit in order to view or reference it later. Listing 3.3 shows an exemplary source control commit log snippet.

<table>
<thead>
<tr>
<th>Revision</th>
<th>Author</th>
<th>Date/Time</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>r216</td>
<td>kkolinko</td>
<td>2011-10-11 10:42:45 +0000</td>
<td>1</td>
</tr>
</tbody>
</table>

```
Modified:
M org/apache/jasper/Constants.java
M org/apache/jasper/JasperException.java

```

Listing 3.3: Source control commit log snippet.

Similar to Java source code, we now explain how to publish source control commit log as RDF dataset based on the principles that are outlined in the beginning of this section.

P1 Identify entities that exists in a software repository

In order to represent source control commit logs in RDF according to the Linked Data principles, we first identify different entities that exists in a source control commit log. Therefore, we consider revision no., source code and developer as entities in the source control commit log.

P2 Assign globally unique identifiers to these entities

After identifying entities in a source control commit log, we assign globally unique identifiers to these entities. In particular, we assign HTTP URIs to these entities so that useful information about each entity can be provided upon request. As an example, we present most commonly found entities in a source control commit log (based on Table 1.2) and their exemplary HTTP URI patterns in Table 3.4.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
<th>Example URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>revision no.</td>
<td>r216</td>
<td><a href="http://example.org/tomcat/svn/216">http://example.org/tomcat/svn/216</a></td>
</tr>
<tr>
<td>developer</td>
<td>kkolinko</td>
<td><a href="http://example.org/tomcat/author/kkolinko">http://example.org/tomcat/author/kkolinko</a></td>
</tr>
</tbody>
</table>

Table 3.4: URI patterns for different types of entities found in a source control commit log.

It is likely that different software projects hosted on a versioning system have similar revision numbers, therefore, to assign globally uniquely identifier to a source control commit log, we use project name in the URI (e.g., we have used “tomcat” in the URIs in Table 3.4) in order to avoid conflicting URIs. As a software developer often work on multiple software projects, therefore, we also use project name in the URI of a software developer in order to keep track of the activities.
of a software developer within a particular software project. Note that the domain name (i.e., example.org) used in the URI patterns listed in Table 3.4 is for illustration purposes only and must be replaced by the domain name of a data publisher.

3 Describe entities and relationships between them using RDF vocabularies

Once the HTTP URIs are assigned to different entities in a source control commit log, the next step is to define the relationships between them. A source control commit log usually contains information about the modifications made to a file or set of files by a specific author in a certain period of time. Therefore, we can define the relationship between the files committed on a source control repository and the author who has made that particular commit. Taking into account the example of a source control commit log shown in Listing 3.3, we can say that "org/apache/jasper/JasperException.java" is modified by an author named "kkolinko". This statement of relationship is, in essence, a link between entities that can be easily expressed explicitly using an RDF statement. In order to show how RDF transformation of a source control commit log may look like, we provide an example of RDF representation of a source control commit log snippet (based on Listing 3.3) in Listing 3.4.

Listing 3.4: RDF representation of a source control commit log based on Listing 3.3.

As mentioned earlier, we assigned URIs to entities that exist in a source control commit log and further defined the association between them. In Listing 3.4, the relationship between a source control commit log and the set of files committed in that particular commit log is represented as RDF statement using baetle:modified property. It tells that a source code file (e.g., http://svn.apache.org/repos/asf/tomcat/trunk/java/org/apache/jasper/Constants) was modified in a commit log revision number "216" (represented in the form of URI as: http://example.org/tomcat/svn/216). Similarly, we have defined the relationship between source control commit log and software developer who have made that particular commit using baetle:author property. Moreover, we have used several other properties to describe meta information (such as summary and timestamp) of revision number “216”, as shown in Listing 3.4.

3.2.3.3 Publishing Bug Repository as RDF Dataset

Bug repository is primarily used to manage software defects in a software project. Moreover, it is used to set milestones and to develop new features for a software project. A bug report usually contains information about the software project, software developer responsible for fixing the bug,
software developer who reported the bug, information about reproducing the bug, priority of the bug etc. Listing 3.5 shows snippet of a bug report taken from the Apache Tomcat bug repository.

Listing 3.5: Bug report snippet taken from the Apache Tomcat project.

Similar to previous software artifacts, we now explain how to publish bug reports as RDF dataset based on the principles that are outlined in the beginning of this section.

**P1 Identify entities that exists in a software repository**

In order to represent bug information in RDF according to the Linked Data principles, we first identify different entities that exists in a bug repository. The two most important things that exists in a bug repository are the bug reports itself and the software developers who have reported, assigned and commented on these bug reports. Thus, we consider bug report ID and software developer as entities.

**P2 Assign globally unique identifiers to these entities**

Similar to previous approaches, we assign HTTP URIs to these entities so that useful information about each entity can be provided upon request. As an example, we present most commonly found entities in a bug report (based on Table 1.2) and their exemplary HTTP URI patterns in Table 3.5.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
<th>Example URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>bug id</td>
<td>51587</td>
<td><a href="http://example.org/tomcat/bug/51587">http://example.org/tomcat/bug/51587</a></td>
</tr>
<tr>
<td>developer</td>
<td>knst.kolinko</td>
<td><a href="http://example.org/tomcat/author/knst.kolinko">http://example.org/tomcat/author/knst.kolinko</a></td>
</tr>
</tbody>
</table>

Table 3.5: URI patterns for different types of entities found in a bug report.

Similar to source control commit logs, it is likely that different software projects hosted on different bug repositories have similar bug IDs, therefore, we use project name in the URI (e.g., we have used “tomcat” in the URIs in Table 3.5) in order to make them globally unique. As a software developer often work on multiple software projects, therefore, we also use project name in the URI of a software developer. For illustration purposes, we have used example.org as a domain name in the URI patterns listed in Table 3.5 and must be replaced by the domain name of a data publisher.

**P3 Describe entities and relationships between them using RDF vocabularies**

https://issues.apache.org/bizfinder/bugzilla.cgi?ctype=xml&id=51587
Once the HTTP URIs are assigned to different entities in a bug report, the next step is to define the relationships between them. A bug report usually contains information about the software defect or feature, the developer who reported the bug and the developer who is assigned that particular bug report. Therefore, we can define the relationship of a bug report to the software developer who has reported the bug and the software developer who is assigned to fix that bug report. Taking into account the example of a bug report shown in Listing 3.5, we can say that bug report "51587" is reported by "knst.kolinko" and "dev" is assigned to fix that bug report. These statements of relationships contained inside a bug report are, in essence, links between different entities that can be easily expressed using RDF statements. In order to show how RDF transformation of a bug report may look like, we provide an example of RDF representation based on the bug report snippet (cf. Listing 3.5) taken from the Apache Tomcat project in Listing 3.6.

```xml
@prefix baetle: <http://baetle.googlecode.com/svn/ns/#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://example.org/tomcat/bug/51587> a baetle:Issue;
  baetle:assigned_to <http://example.org/tomcat/author/dev>;
  baetle:reporter <http://example.org/tomcat/author/knst.kolinko>;
  baetle:summary "Implement status and uptime commands";
  baetle:hasState "";

<http://example.org/tomcat/author/dev> a foaf:Person;
  foaf:accountName "dev";
  foaf:name "Tomcat Developers Mailing List".

<http://example.org/tomcat/author/knst.kolinko> a foaf:Person;
  foaf:accountName "knst.kolinko";
  foaf:name "Konstantin Kolinko".
```

Listing 3.6: RDF representation of a bug report based on Listing 3.5.

We have assigned URIs to entities that exist in a bug report and further defined the association between them. In Listing 3.6, the relationship between a bug report and the reporter is represented as RDF statement using baetle:reporter property. It tells that a software developer named “knst.kolinko” (represented in the form of URI as: `http://example.org/tomcat/author/knst.kolinko`) is the reporter of bug report "51587" (represented in the form of URI as: `http://example.org/tomcat/bug/51587`). Moreover, we have used several other properties to describe meta information (such as summary, priority, creation date, last updated etc.) of bug report "51587", as shown in Listing 3.6.

### 3.2.3.4 Publishing Emails as RDF Dataset

Emails are primarily exchanged among software developers to discuss technical software project related issues. It is also being used by software developers to keep themselves up to date on what co-developers are doing. An email usually contains information about: person who sent the email, recipient of the email, date and time when the email was sent, subject and contents of the email etc. Listing 3.7 shows snippet of an email taken from the Apache Tomcat mailing list archive.
In the following, we now explain how to publish an email as RDF dataset based on the principles that are outlined in the beginning of this section.

**P1** Identify entities that exists in a software repository

In order to represent email in RDF, we first need to identify the different entities that exists in an email. Similar to a bug report, two most important entities that exists in an email are email IDs and software developers who send emails or reply to those emails. Therefore, we consider email ID and software developer as entities.

**P2** Assign globally unique identifiers to these entities

After identifying entities in an email, we assign HTTP URIs to these entities so that useful information about each entity can be provided upon request. As an example, we present most commonly found entities (based on Table 1.2) and their exemplary HTTP URI patterns in Table 3.6.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
<th>Example URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>email id</td>
<td>42715</td>
<td><a href="http://example.org/tomcat/email/42715">http://example.org/tomcat/email/42715</a></td>
</tr>
<tr>
<td>developer</td>
<td>knst.kolinko</td>
<td><a href="http://example.org/tomcat/author/knst.kolinko">http://example.org/tomcat/author/knst.kolinko</a></td>
</tr>
</tbody>
</table>

Table 3.6: URI patterns for different types of entities found in an email.

As a software developer often contribute to multiple software projects, therefore, we use project name in the URI (e.g., we have used “tomcat” in the URIs in Table 3.6) of a software developer in order to keep track of the activities of a software developer within that particular software project. The domain name (i.e., example.org) used in the URI patterns listed in Table 3.6 is for illustration purposes only and must be replaced by the domain name of a data publisher.

**P3** Describe entities and relationships between them using RDF vocabularies

Once the HTTP URIs are assigned to the entities in an email, the next step is to define the relationships between them. As a software developer usually send an email to an intended list of recipients, therefore, we can define the relationship of an email to its sender and its intended receiver. Taking into account the example of an email shown in Listing 3.7, we can say that an email with a message ID (i.e., 42715@mail.gmail.com) is sent by a software developer named “Konstantin Kolinko”. This statement of relationship inside an email is, in essence, a link between entities. As mentioned earlier, RDF allows us to define these kind of relationships explicitly and further link it to other entities. In order to show how RDF transformation of an email may look like, we provide an example of RDF representation of an email snippet (based on Listing 3.7) in Listing 3.8.
As mentioned earlier, we assigned URIs to entities that exist in an email and further defined the association between them. In Listing 3.8, the relationship between an email and its sender is represented as RDF statement using email:from property, which tells that “Konstantin Kolinko” (represented in the form of URI as: http://example.org/tomcat/author/knst.kolinko) is the sender of an email (represented in the form of URI as: http://example.org/tomcat/email/42715). Moreover, we have used several other properties to describe meta information (such as subject, date, body of the email etc.) of an email, as shown in Listing 3.8.

In this section, we have discussed in detail with supporting examples to convert or represent software artifacts as RDF datasets. We have developed custom written scripts, which makes it easier to automatically publish RDF datasets for different software repositories at large scale. Once the RDF datasets are published, the next step is to interlink these RDF datasets where necessary. In the next section, we look into the information contained inside these software repositories and discuss how similar or related software artifacts can be interlinked across software repositories, based on our published RDF datasets.

3.2.4 Link Discovery among Software Repositories

As mentioned earlier, the different software repositories (such as source control systems, bug repositories, mailing lists etc.) used by a software project contains valuable information. For example, software developers discuss bugs or feature requests often on the project mailing list or discussion forum. Source code related changes are also discussed on the project mailing list. In the open source software settings, source code patches are submitted directly to the bug repository by the contributors, which contains fix for a particular bug. The reason is, contributors don’t have commit rights to the source control repository.

Moreover, source control repositories not only contains latest version of the source code but also the authorship information. Further, software developers quite often mention bug IDs while com-
miting source code changes to the source control repository. Hence, relevant information about a software artifact (e.g., source code, bug, software developer etc.) is distributed across different software repositories hosting a software project as shown in Figure 3.6. Such valuable information needs to be identified and stored in a manner in order to facilitate the study of evolution of a software project. Linking software artifacts across different software repositories (cf. Figure 3.6) helps in improving the quality of information and allow practitioners to have a complete view on the software project from different aspects. Thus, we achieve integration between software repositories by applying different heuristics on the extracted information which is discussed next.

Figure 3.6: Link discovery among software artifacts contained inside different software repositories hosting a software project. We show that similar software artifacts exist across different software repositories and forms an implicit connection between them, which is represented through dotted lines.

3.2.4.1 Interlinking Source Control Commit Logs to Source Code and Bugs

A source control commit log contains information about source code files that are modified, deleted or added. Often, bug IDs are mentioned by software developers in the summary of a commit log. Therefore, it is important to identify source code & bug IDs and add RDF links to their respective RDF resources while representing a commit log in RDF.

We use simple text search mechanism to extract certain phrases commonly used by software developers such as, “fixes https://issues.apache.org/bugzilla/show_bug.cgi?id=51587” or “fix for TOMCAT-51587”, in the summary of a source control commit log (e.g., see line #11 of Listing 3.4). When a bug ID is detected, we add an RDF link (i.e., RDF statement) using baetle:fixes property in order to interlink source control commit log to that particular bug report as shown in Listing 3.9 (line #4).

We also interlink the source control repository URLs associated with source code (e.g., see lines #6–7 of Listing 3.4) to the meta information of that particular source code using an owl:sameAs
property as shown in Listing 3.10 (line #13 & #16). In Listing 3.9 and Listing 3.10, baetle:fixes and owl:sameAs properties show interlinking of a source control commit log (i.e., #216) to a bug report (i.e., #51587) and Java source code (e.g., org/apache/jasper/JasperException).

Interlinking source control repository to the source code and bug repository enables software stakeholders to not only retrieve meta information of a particular commit log but also enables them to query any further information about the source code that has been modified or bug report that has been fixed in that particular commit.

3.2.4.2 Interlinking Emails to Source Control Commit Logs and Bugs

Bugs, feature requests and source code are often discussed among software developers on the project mailing lists. Therefore, we exploit such information by processing email contents and add RDF statements based on the type of artifacts identified in the email. For example, if we identify a bug ID, source code filename or a source control commit ID in the content of an email, then we add an RDF statement using rdfs:seeAlso property in order to link the email to the identified software artifact. Furthermore, we add another RDF statement using ld2sd:mentioned property that describe the type of software artifact (e.g., Bug, SVNCommit, JavaSource) identified in the email.

Software development teams often configure email notifications to the project mailing lists whenever a bug report is updated/created on the bug repository and whenever a commit is made to the source control repository. These notifications are configured in order to keep software developers up-to-date on what is happening in different software repositories relevant to a software project. Looking into the mailing list archives of some Apache projects, we found that the conven-
tions used to notify on the mailing list about a source control commit log is something like, “svn commit: r216”, and is usually found in the subject header of an email, where r216 represents the source control commit log (i.e., #216). While parsing the contents of an email, if a source control commit notification is detected, we add an RDF statement using rdfs:seeAlso property in order to interlink the source control commit log to that particular email as shown in Listing 3.11 (see line #11).

Listing 3.11: An interlinking example connecting an email and source control commit log.

```plaintext
@prefix : <http://example.org/tomcat/email/> .
@prefix ld2sd: <http://vocab.deri.ie/ld2sd#> .
@prefix sioct: <http://rdfs.org/sioc/types#> .
@prefix email: <http://simile.mit.edu/2005/06/ontologies/email#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dct: <http://purl.org/dc/terms/> .

:20090704180657967E823888C5 a sioct:MailMessage;
   dct:subject "svn commit: r216 - /tomcat/current/tc5.5.x/STATUS.txt" ;
   email:from <http://example.org/tomcat/author/kkolinko> ;
   rdfs:seeAlso <http://example.org/tomcat/svn/216> ;
   ld2sd:mentioned "SVNCommit".
```

Notification of a bug report on the mailing list depends on the type of bug repository used for a software project. For example, bug reports of APACHE projects are hosted on two different bug repositories: BUGZILLA and JIRA. Our study on the email notifications based on both bug repositories observed that the convention used by BUGZILLA in the subject header of an email is something like, “DO NOT REPLY [BUG 123]...”. On the contrary, the convention used by JIRA in the subject header of an email is something like, “[jira] Created: (X-123)...”, where X-123 represents the bug report number #123 for APACHE PROJECT X. While parsing the contents of an email, if a bug report notification is detected, we add an RDF statement using rdfs:seeAlso property in order to interlink the bug report to that particular email as shown in Listing 3.12 (see line #11).

Listing 3.12: An interlinking example connecting an email and a bug report.

```plaintext
@prefix : <http://example.org/tomcat/email/> .
@prefix ld2sd: <http://vocab.deri.ie/ld2sd#> .
@prefix sioct: <http://rdfs.org/sioc/types#> .
@prefix email: <http://simile.mit.edu/2005/06/ontologies/email#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dct: <http://purl.org/dc/terms/> .

:0249573rds2 a sioct:MailMessage;
   dct:subject "DO NOT REPLY [BUG 51587]..." ;
   email:from <http://example.org/tomcat/author/knst.kolinko> ;
   rdfs:seeAlso <http://example.org/tomcat/bug/51587> ;
   ld2sd:mentioned "Bug".
```

In Listing 3.11 and Listing 3.12, the rdfs:seeAlso property interlinks an email (email ids are 20090704180657967E823888C5 and 0249573rds2) to the source control commit log #216 and bug report #51587. This enables software stakeholders to not only retrieve meta information of an email but also enables them to query further information about a bug report or source control commit log, hence enabling an easy navigation of information across different software repositories.
3.2.5  Link Generation between Software Repositories: Preliminary Results

In order to identify the rich amount of implicit connections that exists among different software repositories and how it can be made explicit through applying Linked Data methodology, we apply our approach to some randomly selected APACHE projects. The reason of choosing APACHE projects is that the repositories of these projects are on the Web and are available to download (i.e. mailing list archives, bug reports, commit logs etc.). Once the information contained inside different software repositories of a software project is available as Linked dataset, it can be easily exploited to address different use cases often found in the software engineering domain.

We gathered data from different software repositories of 5 randomly selected APACHE projects. The range of data we used for each project is listed in Table 3.7. We selected the mailing list data from the date source control commit logs are available for that particular APACHE project. However, we used the bug repository data from the beginning of each APACHE project because bugs are quite often reopened or reproduced once they have been fixed. Table 3.8 gives an overview on the raw data sources that are extracted from different software repositories of each APACHE project.

<table>
<thead>
<tr>
<th>APACHE Projects</th>
<th>Source Control</th>
<th>Mailing list</th>
<th>Bug Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Maven</td>
<td>2003 - 2010</td>
<td>2003 - 2010</td>
<td>2002 - 2010</td>
</tr>
<tr>
<td>Apache Lucene</td>
<td>2001 - 2010</td>
<td>2001 - 2010</td>
<td>2001 - 2010</td>
</tr>
</tbody>
</table>

Table 3.7: APACHE projects data range.

We applied our approach on the raw data contained inside different software repositories in order to produce RDF datasets, as discussed in Section 3.2.3. After publishing the information contained inside different software repositories as RDF, one has RDF datasets; the number of triples (RDF statements) generated based on the data contained inside each software repository for different APACHE projects is shown in Table 3.9.

\( a \) http://maven.apache.org/
\( b \) http://ant.apache.org/
### Table 3.8: An overview of raw data sources extracted from Apache projects.

<table>
<thead>
<tr>
<th>Software Repository</th>
<th>Apache Tomcat</th>
<th>Apache Maven</th>
<th>Apache Ant</th>
<th>Apache Solr</th>
<th>Apache Lucene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Source Code (#files)</td>
<td>1,027</td>
<td>830</td>
<td>827</td>
<td>925</td>
<td>1,316</td>
</tr>
<tr>
<td>Bug Repository (#bugs)</td>
<td>7,149</td>
<td>3,694</td>
<td>5,213</td>
<td>2,129</td>
<td>2,220</td>
</tr>
<tr>
<td>Mailing List (#emails)</td>
<td>43,335</td>
<td>181,211</td>
<td>83,692</td>
<td>32,260</td>
<td>57,079</td>
</tr>
<tr>
<td>Commit Logs (#logs)</td>
<td>5,116</td>
<td>8,633</td>
<td>5,888</td>
<td>3,630</td>
<td>5,088</td>
</tr>
</tbody>
</table>

### Table 3.9: Total number of RDF triples generated for each software repository of different Apache projects.

<table>
<thead>
<tr>
<th>LD2SD Dataset</th>
<th>Apache Tomcat</th>
<th>Apache Maven</th>
<th>Apache Ant</th>
<th>Apache Solr</th>
<th>Apache Lucene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Source Code</td>
<td>92k</td>
<td>36k</td>
<td>69k</td>
<td>45k</td>
<td>71k</td>
</tr>
<tr>
<td>Bug Repository</td>
<td>79k</td>
<td>113k</td>
<td>186k</td>
<td>104k</td>
<td>130k</td>
</tr>
<tr>
<td>Mailing List</td>
<td>482k</td>
<td>1918k</td>
<td>873k</td>
<td>381k</td>
<td>632k</td>
</tr>
<tr>
<td>Commit Logs</td>
<td>56k</td>
<td>126k</td>
<td>89k</td>
<td>56k</td>
<td>101k</td>
</tr>
</tbody>
</table>
3.2.5.1 Link Statistics between RDF datasets

Based on the interlinking methodology described earlier in Section 3.2.4, we compute the number of links established between RDF datasets of different software repositories. We compute these results by executing SPARQL queries on the interlinked RDF datasets. For example, links between source control and bug repository RDF datasets are computed by executing a SPARQL query shown in Listing 3.13:

```sparql
PREFIX baetle: <http://baetle.googlecode.com/svn/ns/#>
SELECT count (distinct ?bug)
WHERE {
}
```

Listing 3.13: SPARQL query that counts the number of RDF links between source control and bug repository RDF datasets.

The number of links established across different RDF datasets of each Apache project under consideration is shown in Table 3.10. SPARQL queries that are used to count the number of links between different RDF datasets can be found in Appendix B (Listing B.1, Listing B.2, Listing B.3 and Listing B.4).

Table 3.10 shows the rich amount of information contained inside different software repositories of a software project that has been exploited by our interlinking methodology. By interlinking software artifacts, one has an integrated view on all software repositories of a software project. Such a comprehensive view can be exploited to answer many questions often asked by software developers. Some of them are listed below:

- which source code files were committed together while fixing bug #51587;
- which software developer is making most changes to the package org.apache.jasper;
- how many bugs are fixed by modifying source code file JasperException.java;
- are there any discussions related to source control commit log #216 among software developers on the mailing list;
- who is talking about JasperException.java on the mailing list;
- what was kkolinko’s last week activity on the project.

In the next section, we consider some of these and other related questions while carrying out a user study in order to evaluate the effectiveness of our LD2SD approach.
<table>
<thead>
<tr>
<th>Source Dataset</th>
<th>Target Dataset</th>
<th>Apache Tomcat</th>
<th>Apache Maven</th>
<th>Apache Ant</th>
<th>Apache Solr</th>
<th>Apache Lucene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit Logs</td>
<td>Java Source Code</td>
<td>7,642</td>
<td>18,774</td>
<td>17,701</td>
<td>8,644</td>
<td>20,803</td>
</tr>
<tr>
<td>Mailing List</td>
<td>Bug Repository</td>
<td>13,648</td>
<td>64,205</td>
<td>21,626</td>
<td>17,567</td>
<td>30,686</td>
</tr>
<tr>
<td>Commit Logs</td>
<td>Bug Repository</td>
<td>700</td>
<td>1,611</td>
<td>1,431</td>
<td>1,718</td>
<td>2,018</td>
</tr>
<tr>
<td>Mailing List</td>
<td>Commit Logs</td>
<td>15,493</td>
<td>28,891</td>
<td>19,916</td>
<td>3,749</td>
<td>6,741</td>
</tr>
</tbody>
</table>

Table 3.10: Cross linking between RDF datasets of different software repositories of Apache projects.
3.3 INTEGRATING LD2SD INTERACTION LAYER INTO AN IDE

As discussed in the previous section, the interconnections among software artifacts in the various software repositories are typically not explicit. Software developers nowadays have to perform keyword-based searches on the Web to find examples of source code or need to manually trace discussions about a bug on discussion forums or project mailing list. The attraction of using semantic technologies in order to address this issue is based on the idea of transforming software artifacts into a conceptually organized and interlinked data space, incorporating data from different software repositories [Damljanovic and Bontcheva, 2008].

With “Linked Data Driven Software Development” (LD2SD), we have introduced a Linked Data based, light-weight framework that allows to integrate software artifacts found in different software repositories. The so created interlinked data space enables uniform querying and browsing facilities. In this section, we elaborate on the top-most part of our LD2SD approach, i.e., the interaction layer (cf. Figure 3.4), and demonstrate how the LD2SD-interaction can be integrated into an Integrated Development Environment (IDE), as we have presented elsewhere [Iqbal et al., 2009a]. Furthermore, we evaluate the traditional ways (i.e., manual approach) of finding relevant information about a particular task against automatic approach (i.e., LD2SD approach) in terms of time spent on different tasks by software developers. As such, we design an experiment and aim to provide an answer to the following question:

Q: Does interlinking software artifacts enable software developers in retrieving relevant information about a particular task from different software repositories efficiently?

As studied by [Ko et al., 2006, 2007], software developers look for several kind of information in their day to day software development tasks. Therefore, we evaluate the traditional ways software developers usually use to find relevant information about a particular task, which is often time consuming and compare it with our automated approach (LD2SD approach), i.e., presenting relevant information to software developers within their Integrated Development Environment (IDE).

With the work at hand, we aim at providing a single interface that can be used to explore related information about software artifacts from different software repositories. Therefore, we present an interface that software developers can use as a plugin in their development environments, such as Eclipse IDE, in order to find related information about Java source code, which may exist in bug, mailing list or source control repositories. We provide a reference implementation for Eclipse IDE in this section, however, our approach is not limited to any particular IDE. For example, one can implement similar kind of interface for PHPStorm10 to facilitate PHP software developers given that the source code is php based and transformed to RDF dataset.

3.3.1 Methodology

The proposed methodology to enable software stakeholders to exploit LD2SD datasets is depicted in Figure 3.7 and outlined in the following:

1. Transform the data (i.e., software artifacts) contained inside different software repositories into RDF based on the Linked Data principles, yielding LD2SD datasets;

10 http://www.jetbrains.com/phpstorm/
Figure 3.7: LD2SD interaction setup. The data from different software repositories are published as RDF and interlinked, yielding LD2SD datasets. Later, they are indexed by an indexing engine so that lookup service can be provided on top of interlinked datasets. Finally, deliver related information to the software developers in their IDEs.

2. Use an indexing service, such as Apache Lucene/Solr to index the LD2SD datasets;

3. Develop a lookup service on top of the indexing service in order to enable software stakeholders to perform keyword-based searches over the LD2SD datasets;

4. Deliver related information about a software artifact from different software repositories to the software stakeholders in their IDEs.

3.3.2 Implementation

For the concrete implementation of LD2SD plugin in order to exploit interlinked datasets, we have chosen Eclipse, a popular Java IDE. Software developers spend most of their time in IDEs such as Eclipse, though they often require to access bug repository in order to log bugs or discuss software development issues with fellow software developers on the mailing list. As we aim to offer a cross-platform, extensible solution, we decided to implement the actual interface of the plugin as a Linked Data application [Hausenblas, 2009b] based on HTML and utilizing the Eclipse
internal Web browser. Alternatively, the LD2SD-interaction is also possible via a standalone Web browser.

3.3.2.1 Interaction via Eclipse plugin

To enable software developers to search for software artifacts or to find related information about software artifacts without leaving their development environment, we have implemented an LD2SD plugin for Eclipse. This enables software developer to retrieve related information about entities (e.g., classes, methods, packages etc.) present in the Java source code of a software project. For example, a software developer is interested in related information about a particular Java class. He is interested to know which bugs are previously fixed by modifying that particular Java class or what are the discussions happened among software developers relevant to that particular Java class. Such kind of information can be easily retrieved via LD2SD plugin. One way to trigger the plugin is to right-click on the Java class in the project explorer and select the “Show Related Information” command from the context menu as shown in Figure 3.8.

![Figure 3.8: Triggering the LD2SD plugin via the context menu.](image)

In response, the lookup service return URIs as entry points and issues automatically different SPARQL queries, which is executed on the entire LD2SD datasets in order to retrieve related information about that particular Java class. Later, the results are merged together and present to the software developer in his/her Eclipse IDE as shown in Figure 3.9. For example, in order to retrieve bugs that have been fixed by modifying a particular Java class, a SPARQL query is automatically executed on the source control RDF datasets. We use baetle:modified and baetle:fixes properties in the SPARQL query to retrieve only those commit logs where that particular Java class was committed and that it also fixed a bug as shown in Listing 3.14.
3.3 INTEGRATING LD2SD INTERACTION LAYER INTO AN IDE

3.3.2.2 Interaction via standalone Web browser

Alternatively to the Eclipse plugin, the interaction is possible via a standalone Web browser. Software stakeholders can issue simple keyword-based queries and can navigate the search results concerning related information about that software artifact. For example, a software developer is interested about information relevant to a certain Java package. He query the LD2SD dataset by using a Web browser. Typically, the software developer will enter, say, a software package name and the lookup service returns all Java classes belonging to that software package as shown in Figure 3.10.

Additionally, the software developer can browse for related information about a certain Java class by clicking the “Related Information” link displayed next to it, which triggers the same phenomena as described in the previous subsection and further shown in Figure 3.11. The Web interface also allows software stakeholders to search for other software artifacts (e.g., bugs, commit logs etc.). In response, it returns the searched software artifact and further enables the software stakeholder to browse any related information about that particular software artifact as we have shown already through the example of Java package (cf. Figure 3.11).
3.3.3 Initial Evaluation

In order to evaluate our LD2SD approach, we prepared test LD2SD datasets by taking into account different software repositories (bug, source control, source code and mailing list repository) of a software project. The size of dataset selected for this evaluation is shown in Table 3.11. This evaluation was carried out in August 2009. The main purpose of this evaluation is to discover how software developers perceive our integration approach in contrast to traditional approaches while finding relevant information from different software repositories. We assessed the usability of LD2SD plugin approach via end-user evaluation requiring participants to perform a set of pre-defined tasks. We measured the time took by each participant to complete the tasks manually and then through the LD2SD plugin. Later, we asked questions to the participant about the usability [Dumas and Redish, 1999] of LD2SD plugin in order to perform the tasks. We want to highlight that this evaluation does not provide any statistical analysis of the significance of the achieved results because the goal of our evaluation is to better understand the characteristics of our proposed solution in contrast to traditional approaches.
3.3 INTEGRATING LD2SD INTERACTION LAYER INTO AN IDE

Figure 3.11: Related information about a particular Java class that is queried automatically from different software repositories and displayed in the Web interface.

3.3.3.1 Participants

12 participants took part in the evaluation. The participants had development experience ranging from one to five years with different backgrounds. Table 3.12 outlines the familiarity of the participants with the different software tools/repositories that were used in the evaluation process.

3.3.3.2 Tasks

The participants were asked to carry out a set of tasks, which are outlined in the following:

**Task 1** Identify all emails that mentions a specific Java class.

**Task 2** Identify all bugs that have been fixed by modifying a specific Java class.

**Task 3** Identify all software developers that are working on a specific Java package.

**Task 4** Identify all emails that mentions a specific Java package.

**Task 5** Identify all bugs that belongs to a specific Java package.
<table>
<thead>
<tr>
<th>Software Artifact</th>
<th>Number of Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code</td>
<td>30</td>
</tr>
<tr>
<td>Bugs</td>
<td>10</td>
</tr>
<tr>
<td>Emails</td>
<td>15</td>
</tr>
<tr>
<td>Commit logs</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 3.11: Size of dataset used for evaluation.

<table>
<thead>
<tr>
<th>No.</th>
<th>Software Artifact</th>
<th>Familiarity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eclipse IDE</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>Bug Repository</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>Discussion forums/Mailing list</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>Subversion plugin for bug tracking system</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Mylyn plugin for Eclipse IDE</td>
<td>45</td>
</tr>
</tbody>
</table>

Table 3.12: Familiarity of the participants with the software tools/repositories that were used in the evaluation.

For Task 2 and Task 5, we installed a subversion plugin for the bug repository\(^\text{11}\)^. The subversion plugin displays commit log information in a tab on the associated bug report. In order to enable the plugin to display a commit log relevant to a particular bug report, the commit log summary must contain the bug ID (e.g., “\textit{fix for LUCENE-2828}”).

3.3.3.3 Procedure

All participants performed their evaluation on the same machine. Machine used for evaluation has a 2.60GHz Intel Core processor, 4GB RAM and 250GB hard disk running on Windows Vista operating system. Each evaluation lasted approximately 28 minutes. The participants were given introduction about the entire evaluation process. Then, they used the system for 10 minutes to get themselves familiar with the project and different software tools/repositories involved in the evaluation process. After that, they carried out the 5 tasks required for this evaluation. Finally, the participants were asked to fill in a questionnaire and provide feedback after finishing the tasks.

We conducted our evaluation in two phases:

**MANUAL APPROACH**

In the first phase, we gave participants access to the Eclipse IDE containing source code of the project, bug repository, source control and mailing list. The participants searched through emails, traversed bug reports and searched source code files in order to carry out each task.

**PLUGIN APPROACH**

In the second phase, we asked participants to carry out the same tasks by using the LD2SD plugin for Eclipse IDE.

\(^{11}\) https://marketplace.atlassian.com/plugins/com.atlassian.jira.plugin.ext.subversion
3.3.3.4 Results

During the first phase, we observed that participants used different heuristics to list the results of each task. After the second phase, we asked the participants to compare the results of both approaches. Some participants found inconsistencies in their collected results for some tasks during both phases. After cross checking, they found that they failed to collect certain results during the first phase (i.e., manual approach) while carrying out those tasks. Further, participants apparently had difficulties going through each bug report in order to identify corresponding bug entries in a source control repository. After the evaluation, we asked the participants to answer a set of yes/no questions relevant to both approaches, which are listed in Table 3.13.

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the LD2SD plugin useful to discover related information?</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Does our approach adds value compared to the usual exploration of related information?</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Is the design and layout of the plugin suited enough for usage?</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Does the integration of software artifacts as an Eclipse plugin offer any advantage?</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.13: Participants feedback after using the LD2SD plugin for Eclipse IDE.

Q: Does interlinking software artifacts enable software developers in retrieving relevant information about a particular task from different software repositories efficiently?

We measured the time spent on each task during both phases of the evaluation and plotted the results based on the average time each participant spent in carrying out the tasks as shown in Figure 3.12.

![Figure 3.12: Average time spent by each participant on performing the tasks manually and through the LD2SD plugin.](image)

The graph clearly shows that using LD2SD based approach, participants spent less time in finding relevant information about a software artifact in contrast to the manual approach, which turns out to be time consuming. However, we don’t see a significant difference in terms of time spent by participants while carrying out task T3. The reason is, participants...
were able to quickly see the author tags listed in the javadocs (e.g., see line#8 in Listing 3.1) of the source code files of a software package by opening them simultaneously in the Eclipse IDE. Moreover, most of the time spent by participants during task T3 was to write down the software developer names so that they could verify the results of manual approach against automated/plugin approach later.

Finding relevant information about software artifacts become more difficult and time consuming if software developers have to deal with large amount of data in software repositories. Hence, we do require automated approaches in order to help software developers in their day to day software development tasks.

3.3.3.5 Participants Satisfaction

In the last phase of the evaluation, we asked participants to fill out a satisfaction questionnaire. Majority of the participants found our approach of extracting related information and presenting it in an integrated manner inside Eclipse IDE interesting and useful. One of the participants wrote:

“... single point access within the Eclipse IDE seems a natural tool to use. It provides information much faster than accessing individual sources. The integrated view is very convenient”

Participants also liked the approach of interlinking Java source code to the source control commit logs and bugs using Linked Data principles. They were able to answer questions such as: (1) who has fixed a particular bug and which source files have been modified in fixing that bug, (2) who should I talk to, and (3) who has been talking about a particular bug on mailing list. A comment from one of the participants highlight this:

“... it saves time for a software developer and provides a unique interface to have a look at all relevant information in a single view ... all relevant information is available on a single click ... the idea looks promising, the tool would be more useful as it evolves and add features”

Based on the feedback obtained from the participants and the time they spent on each task during both approaches (cf. Figure 3.12), we can clearly see that our approach was perceived to be easy and efficient in carrying out different tasks. However, the evaluation also helped us in identifying the limitations of LD2SD plugin. Some comments of the participants on the LD2SD plugin interface were:

“... interface is relatively small, might be an usability issue for large amount of data.”

“... would be interesting to see how well it works with large amount of data–may be difficulties with presentation”

Participants also showed interest in using LD2SD plugin, especially in the case when they had to search information within thousands of emails and hundreds of bug reports for a particular software project. A comment from one of the participants highlight this:

“... I would definitely use the tool once it is available”
3.4 LESSONS LEARNT AND DISCUSSION

In this chapter, we introduced a Linked Data based approach to integrate data from different software repositories of a software project. We argue that Linked Data is a better way to achieve integration between different software repositories. We proposed to transform data from different software repositories into Semantic Web standard data model i.e., RDF and explained our approach with examples to show how the RDF datasets from each software repository may look like. Once the RDF datasets are made available, we explained our interlinking approach to demonstrate integration between different software repositories. Furthermore, we applied our interlinking approach to some selected Apache projects in order to demonstrate the real value of software repositories information integration.

Having interlinked RDF datasets at hand, software developers can easily explore related information from different software repositories by having an integrated view on the information they seek in their daily use. In order to allow software developers to exploit the linked RDF datasets, we developed an easy-to-use interface (LD2SD plugin) that allows them to retrieve related information from different software repositories for a certain Java class or Java package without leaving their preferred development environment, such as Eclipse IDE. Moreover, we carried out a user study based on the traditional approaches to search for related information from different software repositories against LD2SD plugin approach. The results of the study revealed that the manual approach to search for related information in different software repositories is time consuming in contrast to the automated approach provided by LD2SD plugin.

From a software developer’s perspective who has recently joined a project team, it is extremely important to get himself/herself familiar with the project source code and come up to speed with co-developers as quickly as possible. For example, a newcomer to the development team has different questions in his/her mind [Ko et al., 2007] and try to look for answers to these questions by trolling different software repositories, which is often time consuming. Some of the questions are listed below:

- how many software developers are working on software package X;
- how many bugs are related to software package X;
- who should I ask question regarding the task at hand;
- who is the expert software developer on software package X;
- what are the discussions happened among software developers regarding software package X;
- does my source code changes invoke dependency on other fellow software developers.

In order to find answers to the questions listed above, software developers are required to manually search through different software repositories due to the lack of integration among software repositories. However, our LD2SD approach aims to provide answers to these questions. For example, LD2SD plugin can be used to retrieve related information about a software package by right clicking on it in the project explorer of the Eclipse IDE and selecting the “Show Related Information” from the context menu as shown already in Figure 3.8. In response, a SPARQL query is issued automatically, which first retrieves all source code files that belongs to that particular software package. Then, it automatically query the source control RDF datasets to extract bug IDs.
that has been fixed by modifying those source code files. Later, a list of software developers working on these source code files are queried in order to identify the authorship information from the source control repository RDF dataset. Further, discussions happened among software developers relevant to these source code files are extracted by querying the mailing list RDF dataset. These queried are triggered automatically on a single click by the user. The collected information from different software repositories are later merged and presented to the software developer within the Eclipse IDE, as we have shown already in Figure 3.9.

In order to demonstrate the value of retrieving related information from different software repositories using LD2SD plugin, we run our plugin to extract high level information of different software packages of Apache Tomcat project. The results of some software packages are shown in Table 3.14, which shows that a good number of software developers has worked on the org.apache.jasper.compiler software package in contrast to other software packages under consideration and also this software package appears to be the most buggy software package/module in Apache Tomcat project.

<table>
<thead>
<tr>
<th>Software Module</th>
<th>Bugs</th>
<th>SVN Commits</th>
<th>Discussions</th>
<th>Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.jasper.compiler</td>
<td>45</td>
<td>274</td>
<td>257</td>
<td>21</td>
</tr>
<tr>
<td>org.apache.catalina.startup</td>
<td>39</td>
<td>237</td>
<td>219</td>
<td>7</td>
</tr>
<tr>
<td>org.apache.catalina.core</td>
<td>36</td>
<td>344</td>
<td>373</td>
<td>7</td>
</tr>
<tr>
<td>org.apache.coyote.http11</td>
<td>21</td>
<td>205</td>
<td>176</td>
<td>3</td>
</tr>
<tr>
<td>org.apache.catalina.connector</td>
<td>20</td>
<td>198</td>
<td>194</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.14: Relevant information about some selected Apache Tomcat software packages which exists in different software repositories and is automatically retrieved using LD2SD plugin.

In this chapter, we have presented “Linked Data Driven Software Development” (LD2SD), a lightweight Linked Data based methodology to turn software repositories into Linked Data. We have discussed in detail how software artifacts from different software repositories can be transformed into Linked Data and further demonstrated how the interlinked information can be consumed by software stakeholders. We have shown the value of interlinked software artifacts through easy to use interfaces although we emphasize that the underlying interconnected data possesses the real value and therefore its exploitation is not dependent on any visual interface.

Software developers use software repositories to support collaboration in distributed software development. Thus, they adopt different and sometimes multiple identities for these software repositories. Hence, we are required to correctly identify and interlink different and multiple identities of a software developer, which he/she uses while interacting with different software repositories of a software project as discussed in the next chapter.
SOFTWARE DEVELOPER IDENTIFICATION IN SOFTWARE REPOSITORIES

In previous chapter, we introduced “Linked Data Driven Software Development” (LD2SD), a lightweight Linked Data based methodology to relate data across software repositories explicitly and unambiguously. We showed how different software repositories can be integrated to each other by establishing links among their underlying software artifacts. The interlinked data sets can be used for uniform querying and browsing facilities. Therefore, we showed how the interlinked datasets are consumed by software developers using an Eclipse plug-in or standalone Web browser. In the previous chapter, we discussed in detail interlinking different software artifacts but did not discuss modeling and integration of software developer’s related information that also exists in different software repositories. Software developers often adopt different identities for different software repositories and sometimes multiple identities for the same software repository. Hence, there is a need to correctly identify and interlink different identities of a software developer. By interlinking multiple identities of a software developer, we are able to trace software developer activity across different software repositories of a software project.

Therefore, in this chapter we focus on identifying and interlinking multiple IDs (i.e., identities) of a software developer that he/she uses while interacting with different software repositories of a software project. We investigate different types of IDs, which software developers use for different software repositories and present our methodology to identify and interlink them. We present different use case scenarios, which are realized by interlinking multiple identities of a software developer.

The remainder of this chapter is structured as follows:

- In Section 4.1, we discuss the issues related to software developers that often use multiple IDs while dealing with different software repositories of a project. Further, we highlight the needs of interlinking multiple IDs of software developers and outline some research questions.

- Next, in Section 4.2, we evaluate existing approaches for identify multiple IDs of software developers. We further investigate if existing approaches are sufficient enough by applying it to our test dataset.

- After inspecting the results of existing approaches, we present our approach to identify and interlink multiple IDs of software developers in Section 4.3. We present our approach to interlink multiple IDs of software developers not only within different software repositories of a software project but also across different software projects.

- In Section 4.4, we evaluate our proposed approach by applying it on different software repositories hosting a software project as well as across software repositories of 5 different Apache projects. We further compare the interlinking results that are obtained based on our approach against existing approaches.
• We discuss research questions (outlined in Section 4.1) in detail based on the interlinking results that are achieved using our interlinking approach in Section 4.5.

• Finally, Section 4.6 concludes with final remarks.

4.1 IDENTITIES IN SOFTWARE REPOSITORIES

As discussed in the previous chapter, software developers use tools with underlying software repositories to support collaboration in distributed software development. In order to interact with different software repositories that are part of a software project, software developers usually require to adopt an identity for each repository. For example, they are required to adopt an email address in order to send an email to the project mailing list, adopt an ID to push changes to the source control repository, adopt an ID to report bugs or post comments on bugs in a bug repository etc. Software developers often adopt different IDs for each software repository and sometimes multiple IDs for the same repository [Robles and Gonzalez-Barahona, 2005], while interacting with these software repositories.

The software repositories are designed to help software developers collaborate and coordinate but lacks the feature to manage and merge multiple IDs of a software developer, specially in the case of mailing list and bug tracking repositories. Hence, a software developer can register multiple IDs and use them to collaborate and coordinate with other software developers. For example, Figure 4.1 shows a subset of the social network graph derived from the communications exchanged among software developers on the mailing list of an Apache project.

---

![Diagram of a social network graph](image)

**Figure 4.1**: Software developer communicating using multiple IDs on the mailing list. The white color nodes belong to a software developer (Yonik Seeley), who has used two different email IDs to communicate with others on the software project mailing list.

---

1 We have removed domain names (i.e., excluding everything after "@") attached with email IDs in order to keep the privacy of software developers.
In Figure 4.1, we colored two nodes (i.e., white color) in order to show that both email IDs belong to Yonik Seeley who is the creator of Apache Solr and a member of Apache Software Foundation. It is quite clear that Yonik Seeley has used both email IDs to communicate with other software developers on the mailing list. Looking further into the social structure of both graphs (based on both IDs), we see that there are some IDs that are common in both graphs but most of the IDs are part of different social graphs of Yonik Seeley. In an ideal scenario, either both social graphs need to be merged together or at-least there should exist some sort of information conveying that both IDs belong to Yonik Seeley. Without any information provided about the different IDs used by a software developer, a researcher which carry out social network analysis research [de Sousa et al., 2009; Madey et al., 2002; Meneely et al., 2008] may consider both email IDs as two different software developers because using distinct IDs makes software developers appear as different entities. Furthermore, analyzing activities of a software developer [Christley and Madey, 2007] within a software project might be difficult if a software developer is using different and multiple IDs for each software repository.

Apart from adopting an ID for a software repository, software developers sometimes also mention their names or email addresses in the source code file, which they modify or create in order to fix a bug or implement a new feature for a software project. For example, contributors in open source projects submit source code patches either directly to the project mailing list or to the bug repository. They cannot contribute code directly to the source control repository because they are not invited to be part of the core group of software developers of that particular project [Hassan, 2008]. Such contributors quite often mention their names in the javadoc\(^2\) (specifically in the case of a Java project) of the source code files while modifying them in order to fix a particular bug or adding a new feature to the software project functionality. When the core software developers finally commit changes of contributors to the source control repository, they often mention the name of the contributor who have provided the patch, in the summary of a commit log (e.g., “patch provided by John. Fixes LUCENE-1234…”). Such information is useful for analysis if extracted properly and interlinked to the correct software developer. In the following, we now briefly discuss different types of IDs that software developers use for different software repositories as discussed by [Robles and Gonzalez-Barahona, 2005] and also summarized in Table 4.1:

- In a source code file, software developers appear with many different IDs, such as real life names, email addresses, SVN identifiers and sometimes the combination of real life name and email addresses;
- Software developers usually use multiple email addresses to send emails to the project mailing list. Sometimes, the email headers contain the `<name, email>` pair, which helps to link an email address to a software developer;
- To commit source code to a source control repository, software developers use a separate account on the versioning system;
- Bug repositories require an account associated with an email address in order to log new bugs or post comments on existing bugs.

\(^2\) http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html
Table 4.1: Different types of IDs found in software repositories [Robles and Gonzalez-Barahona, 2005]. It also shows that the software developer IDs in different software repositories are represented in different formats, which makes it difficult to interlink them to each other.

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Identities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Code</td>
<td>Name Surname</td>
</tr>
<tr>
<td>Source Code</td>
<td><a href="mailto:username@domain.com">username@domain.com</a></td>
</tr>
<tr>
<td>Source Code</td>
<td>Name[<a href="mailto:username@domain.com">username@domain.com</a>]</td>
</tr>
<tr>
<td>Source Code</td>
<td>$subversionID</td>
</tr>
<tr>
<td>Mailing List</td>
<td><a href="mailto:username@domain.com">username@domain.com</a></td>
</tr>
<tr>
<td>Mailing List</td>
<td>Name Surname<a href="mailto:username@domain.com">username@domain.com</a></td>
</tr>
<tr>
<td>Source Control</td>
<td>$subversionID</td>
</tr>
<tr>
<td>Bug Tracking</td>
<td>$userID</td>
</tr>
<tr>
<td>Bug Tracking</td>
<td>Name Surname&lt;userID&gt;</td>
</tr>
</tbody>
</table>

4.1.1 Motivation

Using distinct IDs for different software repositories makes software developers appear as different entities. However, the fact that the sets of users differ between the software repositories increases the difficulties of mapping IDs between them. This is a further challenge beside the fact that we have no 1:1 mapping, neither in one software repository nor over different software repositories. Actually, we often encounter an n:m mapping, e.g., several email addresses might belong to the same software developer and multiple software developers might use the same email address. An example of the later is an address like jira@apache.org. Updates of any type on the bug repository are notified to other fellow software developers by automatically sending an email to the mailing list where the email header is of the format, e.g., “Name Surname <jira@apache.org>”, where “Name Surname” refers to the software developer who made changes to a particular bug on the bug repository. While this might be obvious in this case, it poses a significant challenge in automated approaches for identifying multiple IDs of a software developer.

Software developers often contribute to multiple open source projects based on their interests. Therefore, the development activities of software developers are distributed not only within different software repositories of a software project but also across different software projects. It is worth mentioning that there exists an implicit connection between software developers development activities within different software repositories of a software project and also across different software projects. Therefore, we require methods and techniques to correctly identify and interlink different IDs of software developers not only within a software project but also across different software projects. This is a main requirement, among others, to be able to keep track of software developers development activities within different software repositories as well as across different software projects.

In general, the problem is related to identifying different representations of the same real-world object and is known by different terms, such as record linkage, object identification, duplicate detection and so on. The problem of finding duplicate items has been researched in the past in the context of data management and Web applications. For example, the work done by [Ananthakrishna et al., 2002; Chaudhuri et al., 2005; Monge, 2000; Monge and Elkan, 1997; Tian et al., 2002] focused on detecting duplicate records in databases and the work done by [Conrad, 2003; Di Lucca et al., 2002; Ye et al., 2004] focused on detecting duplicate documents and Web pages. In the field
of software development, duplicate detection has been also taken into consideration for different software artifacts that are produced as part of a large software project. Researchers have done considerable work in order to detect duplicate bug reports [Jalbert and Weimer, 2008; Runeson et al., 2007] and duplicate source code [Ducasse et al., 1999; Komondoor and Horwitz, 2001], which exists in a software project in order to reduce the time that software developers spend in triaging the bug reports or fixing duplicate source code.

While research in the duplicate detection domain mostly refers to identifying duplicates in the same data set, the techniques may be mapped to the case of matching over different data sets as it is in the case of matching software developer IDs across different software repositories and software projects.

4.1.2 Questions

We list down some research questions in the following, which we address later based on interlinking multiple IDs of a software developer:

Q1: What is the added benefit of interlinking multiple IDs of software developers?
We investigate if software developers are using multiple IDs within a software repository and if interlinking them provide any added benefits? We validate this on the bug repository dataset by computing the number of bugs reported by software developers using multiple IDs. This will show us a clear picture on the amount of information, which is not connected due to the usage of multiple IDs by software developers.

Q2: What is the ratio of software developer’s existence in multiple software projects?
We investigate the frequency of software developer’s participation in multiple software projects. In particular, we investigate if software developer’s participation have power-law distribution [Schroeder, 1991], i.e., many software developers participate in few software projects and only few software developers participate in many software projects.

Q3: What are the contributions made by software developers in multiple software projects?
As mentioned earlier, software developers often contribute to multiple open source software projects. Therefore, we evaluate to what extent our interlinking methodology helps in tracing software developers development activities across multiple software projects. In particular, we investigate if software developers are highly active in pushing source control commits, reporting bugs or participating in discussions on project mailing list across multiple software projects.

In this section, we have motivated with an example of the usage of multiple IDs by software developers while interacting with different software repositories of a software project. Further, we have outlined different formats in which software developer IDs are represented for different software repositories and presented some research questions that can be answered once multiple IDs of software developers are interlinked. In the next section, we evaluate existing software developer ID identification approaches by applying it on our test dataset and further compute the precision and recall of these approaches.
4.2 Existing Software Developer Identification Approaches

There are only few published works on identifying and relating different IDs that software developers use to interact with different software repositories. [Bird et al., 2006] proposed an approach to produce a list of <name, email address> identifiers by parsing the emails and clustering them. The clustering algorithm to measure the similarity between every pair of IDs is based on string similarity between names, between emails, between names and emails, etc. Two IDs with a similarity measure lying below a pre-defined threshold value are placed into the same cluster. The authors use different approaches to compute the similarity measures between every pair of IDs, some of which we tested on our test dataset to validate the effectiveness of these approaches (cf. Section 4.2.1 and Section 4.2.2).

[Robles and Gonzalez-Barahona, 2005] discusses the problem of software developer identification in general, but the work lacks in details about the heuristics they propose to identify and match the different IDs of software developers. This makes it difficult to validate their approach for solving this problem. The authors propose a technique to build one identity from another by extracting the “real life” name from email addresses, such as nsurname@domain.com, name.surname@domain.com etc. This is an approach based on pre-defined name schemes, a variant that we also evaluate for our case in Section 4.2.3.

In [Iqbal and Karnstedt, 2010], we present results based on the above described approaches gained on data from a representative open source project Apache Tomcat. We executed some experiments to evaluate the methods described in [Bird et al., 2006] and [Robles and Gonzalez-Barahona, 2005]. We gathered data by parsing emails from Apache Tomcat mailing lists starting from 2005 till 2010. For every email containing a written name and the according email address, we extracted the from and to field from the email header to produce a list of <name, email address> pairs. We processed 49,927 emails from the mailing list archives and produced 1261 distinct <name, email address> pairs. We use these results as a test set to validate different techniques that could be utilized to interlink different IDs of a software developer. The reason for using data from a set where we actually know the exact matching is simple: it is the only way to provide some ground truth. It is not possible to extract similar information for other types of IDs. The only, clearly impractical, way is to ask all involved persons for the different IDs they use. However, we strongly believe that the gained results are significant for these other types of IDs as well. Moreover, the methods we tested were proposed for matching email addresses.

To check the effectiveness of each approach, we first used the approach to find a matching software developer name for each email address. Then, we computed two versions of precision \( P \) and recall \( R \) values for each approach using the following equations:

\[
P_1 = \frac{\# \text{email addresses with at least one correct match}}{\# \text{matched email addresses}}
\]

\[
R_1 = \frac{\# \text{email addresses with at least one correct match}}{\# \text{total email addresses}}
\]

\[
P_2 = \frac{\# \text{email addresses with exactly one correct match}}{\# \text{matched email addresses}}
\]

\[
R_2 = \frac{\# \text{email addresses with exactly one correct match}}{\# \text{total email addresses}}
\]
By determining these different types, we gain some interesting additional knowledge. For $P_1$ and $R_1$, we count a hit if we found at least one correct match (if a name and email address matches), i.e., even if we found more incorrect matches. This provides an “optimistic” quality assessment, as the correct match(es) are found. Anyhow, in reality one still has to filter out the wrong matches. Thus, we determine a “pessimistic” quality assessment in $P_2$ and $R_2$. If there exist correct and wrong matches for a given email this is not regarded as a hit.

In order to compute the similarities between software developer names and email addresses or between two email addresses (see Section 4.2.1 and Section 4.2.2), we used the Levenshtein edit distance [Ukkonen, 1985] algorithm, as suggested by [Bird et al., 2006]. The match(es) for an email address are those software developer names with the lowest distance. Clearly, there is no sense in allowing arbitrary large distances, as this mean that one string can be transformed into a completely other one. Thus, we tested three different threshold values:

- the maximal length of both strings
- the minimal length of both strings
- a fixed threshold value of 4

4.2.1 Similarity between Names and Email Addresses

In the first test, we extracted a list of software developer names and email addresses from our test set. Later, we matched each software developer name against all email addresses (excluding the domain after “@”) using the above listed three different threshold values by doing a pairwise comparison, based on Levenshtein edit distance. In order to check the effectiveness of this approach, we computed the precision and recall values for our matched result set by taking into account our test set, which are shown in Table 4.2.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>$P_1$ in %</th>
<th>$R_1$ in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum length</td>
<td>67.1</td>
<td>57.1</td>
</tr>
<tr>
<td>Minimum length</td>
<td>57.4</td>
<td>46.3</td>
</tr>
<tr>
<td>Fix Threshold</td>
<td>54.4</td>
<td>30.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threshold</th>
<th>$P_2$ in %</th>
<th>$R_2$ in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum length</td>
<td>24.3</td>
<td>20.1</td>
</tr>
<tr>
<td>Minimum length</td>
<td>18.4</td>
<td>15.3</td>
</tr>
<tr>
<td>Fix Threshold</td>
<td>48.4</td>
<td>27.2</td>
</tr>
</tbody>
</table>

Table 4.2: Names-email address similarity results.

The results for $P_1$ and $R_1$ are actually quite good, where the maximal-length threshold seems to be the best choice. However, the accuracy is not high enough. Moreover, the values of $P_2$ and $R_2$ show that these methods produce a lot of false matches. As the fixed threshold seems to be the most “stable” in that context, we can learn that high thresholds tend to match very different strings – and thus create more false positives. In general, all methods are not very well suited for practice on a dataset like ours.
4.2.2 Similarity between Email Addresses

Besides names and emails, it is likely that email addresses are textually similar to each other if they belong to the same software developer. Thus, in the second test, we considered only email addresses from our test set. For every email address (excluding the domain after “@”), we computed the pairwise similarities with every other email address (excluding the domain after “@”) and determined match(es) for one email address by choosing those with the lowest distance. This was also suggested by [Bird et al., 2006]. In order to check the effectiveness of this approach, we computed the precision and recall values for our matched result set by taking into account our test set, which are shown in Table 4.3.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>$P_1$ in %</th>
<th>$R_1$ in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum length</td>
<td>3.2</td>
<td>3.2</td>
</tr>
<tr>
<td>Minimum length</td>
<td>3.1</td>
<td>3.0</td>
</tr>
<tr>
<td>Fix Threshold</td>
<td>7.2</td>
<td>2.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threshold</th>
<th>$P_2$ in %</th>
<th>$R_2$ in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum length</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Minimum length</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Fix Threshold</td>
<td>3.5</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 4.3: Email address similarity results.

Clearly, this idea does not work at all for our dataset. The low values show that assuming similarity between emails in order to find matches is an absolutely inappropriate solution for our case.

4.2.3 Matching based on Name Schemes

Most likely, email addresses are built from the “real life” name of a software developer. As suggested by [Robles and Gonzalez-Barahona, 2005], in the third test we tried to identify matching software developer names for email addresses (excluding the domain after “@”) by checking the relation between them based on different name schemes. For example, *aftab.iqbal@deri.org* matches to *Aftab Iqbal* based on the name scheme *name.surname*. Based on our observations on different open source projects, we selected different name schemes that software developers quite often use to build their email addresses: *name.s* (e.g., *aftabi*), *n.surname* (e.g., *aiqbal*), *n.s* (e.g., *ai*), *na.su* (e.g., *a/iq*), *name.surname* (e.g., *aftabiqbal*), *e.surname* (e.g., *biqbal*) and *name* (e.g., *aftab*). Note that the dot in a name scheme is only for illustration purposes – in the actual matching we do not use dots and ignore them in email addresses if present. For every software developer name present in our test set, we built candidate email addresses (without the domain after “@”) using above mentioned name schemes and matched it against email addresses (without the domain after “@”) in our test set. In order to check the effectiveness of this approach, we computed the precision and recall values for our matched result set (for each name scheme) by taking into account our test set, which are shown in Table 4.4.

To our surprise, most of the name schemes result in high precision values, for both types of quality measures. This means that if we find a match with these techniques, it is likely to be a
correct match. However, the in contrast low recall values show that this method is capable of identifying only a handful of all matches. Thus, on its own, it is also not suited for our use case.

### 4.2.4 Conclusion

Based on the results achieved, we can say that all three tested approaches do not prove to be suited for our case. As we have some good results in parts, it seems to be promising to combine these techniques with some more advanced approaches. By inspecting the precision and recall values, we can conclude that combining the methods among themselves is not promising. Hence, we require a more thorough approach, which maximize the chances of identifying multiple IDs of a software developer. Thus, in the following section, we present our simple yet effective approach to complement existing software developer ID identification approaches.

### 4.3 Our Approach

In this section, we describe our approach to identify and interlink different IDs of a software developer found in various software repositories of a software project as well as across repositories of multiple software projects [Iqbal and Hausenblas, 2013]. While transforming software repositories into RDF datasets (cf. Section 3.2.3), we assign URIs to software developers and use different properties to describe a software developer name, account ID or email address. We apply different heuristics to identify and interlink different URIs (which belongs to the same software developer within a software repository as well as across software repositories of multiple software projects) through inserting `owl:sameAs` links, which are explained in the following.

We consider mailing list and bug repository data sources because software developers often use multiple IDs on these software repositories as opposed to source control repository where access
and accounts are controlled centrally. As mentioned by [Bird et al., 2006], most emails contain header of the form:

From: "aftab iqbal" <aftab.iqbal@deri.org>

where “aftab iqbal” is the name of the person associated with the email address, i.e., “aftab.iqbal@deri.org”. For each email, we represent header information in RDF using foaf vocabulary as shown in Listing 4.1 (see lines #11-13). We refer the readers to Section 3.2.3.4 for details on our approach to transform an email to RDF. We use email alias (e.g., aftab.iqbal in the case of aftab.iqbal@deri.org) to build a software developer URI (see Table 3.6 for more information about the usage of URI patterns for a software developer) because email alias can be easily used to distinguish between different software developers.

Listing 4.1: RDF representation of an email.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix sioc: <http://rdfs.org/sioc/types#>.
@prefix email: <http://simile.mit.edu/2005/06/ontologies/email#>.
@prefix : <http://example.org/tomcat/email/>.

:10029029 a sioc:MailMessage ;
email:from :example.org/tomcat/author/aftab.iqbal ;
email:subject "compilation problem" ;
email:body "while compiling the latest source ..."
.
<http://example.org/tomcat/author/aftab.iqbal> a foaf:Person ;
foaf:name "aftab iqbal" ;
foaf:mbox <mailto:aftab.iqbal@deri.org>
.
```

While transforming emails to RDF using our custom written scripts, we found that certain software developers have used either full or part of their names in the email header. For example:

From: "muhammad aftab iqbal" <aftab.iqbal@deri.org>

The above two email headers reveal multiple names of the sender (i.e., aftab iqbal and muhammad aftab iqbal). The software developer URI remains the same in the case of above two exemplary email headers (i.e., http://example.org/tomcat/author/aftab.iqbal), however, the property values are changed (name in that case) as each email header belongs to a different email. Hence, multiple names associated to an email address are extracted to produce a list of <name,id> pairs for a particular software developer, which are later used for matching purposes.

In the case of bug reports, we found that each bug report usually represents software developer information of the form:

```
<assignee username="ai">aftab iqbal</assignee>
<reporter username="jim">jim</reporter>
```

where ai is the ID associated with the person named as aftab iqbal. We represent the software developer information found in each bug report using foaf vocabulary as shown in Listing 4.2 (see lines #11-13). We refer the readers to Section 3.2.3.3 for details on our approach to transform a bug report to RDF. For information about the usage of URI patterns for software developers while transforming a bug report to RDF, we refer the readers to Table 3.5.
After transforming data into RDF, we loaded RDF datasets into SPARQL endpoint. The next step is to identify and interlink multiple IDs of a software developer. From Listing 4.1 and Listing 4.2, we are able to conclude that the software developer “aftab iqbal” has used different IDs (i.e., after.iqbal and ai) while interacting with others on the mailing list and bug repository. We can interlink these two RDF fragments using an owl:sameAs property indicating that both software developer URIs actually refers to the same software developer, as shown in Listing 4.3.

By explicitly interconnecting IDs of a software developer, we are able to trace software developer’s activity on the mailing list as well as bug repository. However, manually identifying and interlinking IDs of software developers is a time consuming task especially when dealing with large amount of data. Hence, we are required to devise an automatic approach for the identification and interlinking of software developer IDs. In the previous section, we applied some existing approaches on test dataset but they do not prove to be suited for our case. Therefore, in the following we present our simple yet effective approach for interlinking software developer IDs.

4.3.1 Preliminaries

Let Authors be a list of distinct software developer URIs found in the dataset:

\[
\text{Authors} = \{A, B, C, D, \cdots\}
\]  

where

\[
A = \text{http://example.org/tomcat/author/aftab.iqbal}
\]  

and

\[
B = \text{http://example.org/tomcat/author/ai}
\]
Let \( \tilde{A} \) and \( \check{B} \) be the distinct <name, email address> pairs for \( A \) and \( B \) respectively:

\[
\tilde{A} = \left( < \text{aftab iqbal}, \text{aftab.iqbal@deri.org} > \right) \tag{4.8}
\]

\[
\check{B} = \left( < \text{aftab iqbal}, \text{ai@example.com} > \right) \tag{4.9}
\]

Let \( L \) be a set, which contains a collection of distinct software developer URIs along with their associated <name, email address> pairs:

\[
L = \left( \begin{array}{c}
(A, \tilde{A}) \\
(B, \check{B}) \\
(C, \check{C}) \\
\vdots
\end{array} \right) \tag{4.10}
\]

4.3.2 Interlinking IDs within a Software Repository

In order to identify multiple IDs of a software developer within a software repository, we first build a list of software developer URIs and their corresponding <name, email address> pairs, hence call it set \( L \) as shown in Equation 4.10. Later, we compare each software developer URI with every other software developer URI in set \( L \) by comparing their respective <name, email address> (for email address, we exclude everything after “@”) pairs. If a match is found, we establish an owl:sameAs link.
between both software developer URIs as shown in Listing 4.3. The pseudocode of our approach is shown in Algorithm 4.3.1.

**Algorithm 4.3.1: ID_LINKING_IN_A_REPOSITORY()**

```plaintext
lstURI ← get distinct software developer URIs
for i ← 0 to lstURI.size() do
  lst ← get IDs and names associated with lstURI[i]
  hashMap.add(devURI[i], lst)

/* match each software developer URI with every other software developer URI in the list based on the <name, emailaddress> pairs */
for i ← 0 to hashMap.size() do
  id1 ← hashMap.getKey(i)
  lst1 ← hashMap.getValue(id1)
  for j ← i + 1 to hashMap.size() do
    id2 ← hashMap.getKey(j)
    lst2 ← hashMap.getValue(id2)
    if matched(lst1, lst2) then
      generateSameAsLink(id1, id2)

procedure matched(lst1, lst2)
for i ← 0 to lst1.size() do
  arr1 ← get IDs and names associated with lst1[i]
  for j ← 0 to lst2.size() do
    arr2 ← get IDs and names associated with lst2[j]
    if arr2[0].equals(arr1[0]) && arr2[1].equals(arr1[1]) then return true
  return false
```

4.3.3 Interlinking IDs across Software Projects

In order to identify similar or different IDs of a software developer across two different software projects, we first build a list of software developer URIs and their corresponding <name, emailaddress> pairs for both software projects. For each software developer URI, we also query the owl:sameAs links (if any) in order to build a comprehensive list of <name, emailaddress> pairs. The owl:sameAs links between multiple IDs of a software developer within a software repository of a software project is computed based on the approach described in Section 4.3.2. Once we have the lists of software developer IDs and their corresponding <name, emailaddress> pairs for both software projects, we then compute similar or different IDs belonging to a software developer across both software projects, as explained in the following.
Let $Prj_A$ and $Prj_B$ (cf. Equation 4.11 and Equation 4.12) be the sets, which contains the collection of distinct software developer URIs along with their associated <name, email address> pairs extracted from the software repositories of Project A and Project B such that:

$$Prj_A = \begin{pmatrix} (A, \hat{A}) \\ (B, \hat{B}) \\ (C, \hat{C}) \\ \vdots \end{pmatrix}$$ (4.11)

$$Prj_B = \begin{pmatrix} (A, \hat{A}) \\ (B, \hat{B}) \\ (C, \hat{C}) \\ \vdots \end{pmatrix}$$ (4.12)

We compare each software developer URI in set $Prj_A$ with every software developer URI in set $Prj_B$ by comparing their respective <name, email address> pairs. We use the full email address (including the domain after “@”) in this case because we are identifying software developer IDs across software projects and it is likely that software developers use same email addresses across different software projects. If a match is found while doing pair-wise comparisons based on <name, email address> pairs for a software developer URI in $Prj_A$ with the <name, email address>
pairs for a software developer URI in PrjB, we establish an owl:sameAs link between both software developer URIs (cf. Listing 4.3). The pseudocode of our approach is shown in Algorithm 4.3.2.

Algorithm 4.3.2: id_linking_across_projects()

prj_A ← get distinct software developer URIs from project A
prj_B ← get distinct software developer URIs from project B

for i ← 0 to prj_A.size()
    lstID ← getSameAsLinks(prj_A[i])
    for j ← 0 to lstID.size()
        lst ← get emails and names associated with lstID[j]
        lstPairs.add(lst)
        hshMap_A.add(prj_A[i], lstPairs)
for i ← 0 to prj_B.size()
    lstID ← getSameAsLinks(prj_B[i])
    for j ← 0 to lstID.size()
        lst ← get emails and names associated with lstID[j]
        lstPairs.add(lst)
        hshMap_B.add(prj_B[i], lstPairs)

/* match each software developer URI of project A with every software developer URI in project B based on the < name, emailaddress > pairs */

for i ← 0 to hshMap_A.size()
    id1 ← hshMap_A.getKey(i)
    lst1 ← hshMap_A.getValue(id1)
    for j ← 0 to hshMap_B.size()
        id2 ← hshMap_B.getKey(j)
        lst2 ← hshMap_B.getValue(id2)
        if matched(lst1, lst2)
            then generateSameAsLink(id1, id2)

procedure matched(lst1, lst2)

for i ← 0 to lst1.size()
    arr1 ← get IDs and names associated with lst1[i]
    for j ← 0 to lst2.size()
        arr2 ← get IDs and names associated with lst2[j]
        if arr2[0].equals(arr1[0]) & arr2[1].equals(arr1[1])
            then return (true)
return (false)

The outcome of our above described matching approach is a set of RDF files in N-TRIPLES\(^3\) format, which contains the software developer URIs having owl:sameAs links between them (e.g., see Listing 4.3). The RDF files are later loaded into the SPARQL endpoint so that it can be used to query development activity of software developers within and across different software projects.

\(^3\) http://www.w3.org/2001/sw/RDFCore/ntriples/
In the next section, we apply our software developer identity matching approach on 5 different Apache software projects. In particular, we apply our approach to identify multiple and different identities of software developers within a software repository as well as across software repositories of different Apache software projects. Moreover, we compare the matching results of our approach against existing approaches that are described in Section 4.2.

4.4 PRELIMINARY EVALUATION

Before we discuss the results of our matching approaches, we list down the Apache projects selected for evaluation. As stated in the previous chapter, we select Apache projects because the information about them are contained inside different software repositories (i.e. mailing list archives, bug reports, commit logs etc.) that are available to download. Therefore, we randomly selected some popular Apache projects for the evaluation of matching approaches described previously. Moreover, we consider data (contained inside different software repositories) from the beginning of each selected Apache project till 2012, as shown in Table 4.5.

<table>
<thead>
<tr>
<th>Apache Projects</th>
<th>Date Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant</td>
<td>2000 - 2012</td>
</tr>
<tr>
<td>Apache Hadoop</td>
<td>2006 - 2012</td>
</tr>
<tr>
<td>Apache Logging</td>
<td>2001 - 2012</td>
</tr>
<tr>
<td>Apache Lucene</td>
<td>2001 - 2012</td>
</tr>
<tr>
<td>Apache Maven</td>
<td>2003 - 2012</td>
</tr>
</tbody>
</table>

Table 4.5: Apache projects data range.

4.4.1 Interlinking IDs within a Software Repository

In order to identify and interlink multiple IDs of software developers that exists in a repository, we apply our interlinking approach (cf. Algorithm 4.3.1) on the mailing list and bug repository of each Apache software project separately. During the matching phase, we found certain IDs that are more generic and multiple software developers are associated to those IDs. We exclude those IDs during the matching phase. Example of those IDs are: jakarta, jakarta-ant, lists, ant-dev, dev, ant, apache, general, hadoop, nutch-dev, log4j, log4j-dev, java-dev, lucene etc. The outcome of our interlinking approach is listed in Table 4.6 and Table 4.7, where the “IDs found” column tells the distinct IDs found with a match and the “links” column tells the total number of owl:sameAs links established between IDs.

<table>
<thead>
<tr>
<th>Apache Projects</th>
<th>No. of software developers</th>
<th>IDs found</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant</td>
<td>2,469</td>
<td>453</td>
<td>320</td>
</tr>
<tr>
<td>Apache Hadoop</td>
<td>1,260</td>
<td>189</td>
<td>118</td>
</tr>
<tr>
<td>Apache Logging</td>
<td>988</td>
<td>105</td>
<td>76</td>
</tr>
<tr>
<td>Apache Lucene</td>
<td>1,506</td>
<td>208</td>
<td>158</td>
</tr>
<tr>
<td>Apache Maven</td>
<td>1,886</td>
<td>340</td>
<td>246</td>
</tr>
</tbody>
</table>

Table 4.6: Matched IDs of software developers based on mailing list repository.
Based on the matching results (cf. Table 4.6 and cf. Table 4.7), we found that the usage of multiple IDs is mostly common on the mailing list than the bug repository except the case of APACHE ANT project. However, it is still important to identify and interlink multiple IDs of a software developer for every software repository. Same interlinking approach can be used to match software developer IDs across different software repositories (i.e., matching mailing list and bug repository software developer IDs) of a software project.

### 4.4.2 Interlinking IDs across Software Projects

In order to identify and interlink IDs of software developers across different software projects, we apply our interlinking approach (cf. Algorithm 4.3.2) to the bug and mailing list repository of APACHE projects. We consider two IDs to be similar if either the <name, email address> pairs of both IDs matches or if only the name or email address of the IDs matches. During the evaluation, we found match for certain software developers who have used single word name (e.g., chris, brian, J etc.). As it is difficult to differentiate if the software developers are same (based on a single word name), therefore, we strictly consider a match if the name consists of at-least two words (i.e., name surname). The results of our cross software project matching approach based on mailing list and bug repository datasets are shown in Table 4.8 and Table 4.9.

The result shows that matching based on either name or email address generates more links in contrast to the <name, email address> approach. Moreover, there are good number of software developers whose IDs are found in multiple software projects. The most prominent among them are the software developers of APACHE ANT and APACHE MAVEN software project (cf. Table 4.8). The reason behind it could be the technical dependencies between both software projects (e.g., reusing software components) or the interest of software developers working on similar kind/category of software projects.

### 4.4.3 Comparison of Software Developer Identification Approaches

We apply few suggested approaches (see Section 4.2) on our dataset in order to validate the effectiveness of these approaches against our approach. We apply these approaches only on the mailing list dataset because we found a high ratio of multiple IDs being used by software developers on the mailing list in contrast to the bug repository dataset. In order to identify multiple IDs of a software developer, we computed similarities between software developer names and email addresses or between two email addresses using three different threshold values as discussed in Section 4.2. The results of both approaches using different threshold values comparing to our approach is plotted in the form of chart shown in Figure 4.2.

<table>
<thead>
<tr>
<th>APACHE Projects</th>
<th>No. of software developers</th>
<th>IDs found</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>APACHE ANT</td>
<td>3,927</td>
<td>143</td>
<td>76</td>
</tr>
<tr>
<td>APACHE HADOOP</td>
<td>1,219</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>APACHE LOGGING</td>
<td>1,131</td>
<td>42</td>
<td>21</td>
</tr>
<tr>
<td>APACHE LUCENE</td>
<td>981</td>
<td>27</td>
<td>15</td>
</tr>
<tr>
<td>APACHE MAVEN</td>
<td>1,577</td>
<td>41</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 4.7: Matched IDs of software developers based on bug repository.
Table 4.8: Matched IDs of software developers across Apache projects based on mailing list repository.

<table>
<thead>
<tr>
<th>ProjectA (no. of developers)</th>
<th>ProjectB (no. of developers)</th>
<th>Mailing List</th>
</tr>
</thead>
<tbody>
<tr>
<td>APACHE ANT (2469)</td>
<td>APACHE HADOOP (1260)</td>
<td></td>
</tr>
<tr>
<td>APACHE ANT (2469)</td>
<td>APACHE LOGGING (988)</td>
<td>24 12 59 43</td>
</tr>
<tr>
<td>APACHE ANT (2469)</td>
<td>APACHE LUCENE (1506)</td>
<td></td>
</tr>
<tr>
<td>APACHE ANT (2469)</td>
<td>APACHE MAVEN (1886)</td>
<td>198 100 365 254</td>
</tr>
<tr>
<td>APACHE HADOOP (1260)</td>
<td>APACHE LOGGING (988)</td>
<td>2 1 6 3</td>
</tr>
<tr>
<td>APACHE HADOOP (1260)</td>
<td>APACHE LUCENE (1506)</td>
<td>86 43 121 82</td>
</tr>
<tr>
<td>APACHE HADOOP (1260)</td>
<td>APACHE MAVEN (1886)</td>
<td>34 17 48 29</td>
</tr>
<tr>
<td>APACHE LOGGING (988)</td>
<td>APACHE LUCENE (1506)</td>
<td>40 20 68 41</td>
</tr>
<tr>
<td>APACHE LOGGING (988)</td>
<td>APACHE MAVEN (1886)</td>
<td>96 51 156 103</td>
</tr>
<tr>
<td>APACHE LUCENE (1506)</td>
<td>APACHE MAVEN (1886)</td>
<td>92 46 134 81</td>
</tr>
</tbody>
</table>

Table 4.9: Matched IDs of software developers across Apache projects based on bug repository.

<table>
<thead>
<tr>
<th>ProjectA (no. of developers)</th>
<th>ProjectB (no. of developers)</th>
<th>Bug Tracking System</th>
</tr>
</thead>
<tbody>
<tr>
<td>APACHE ANT (3927)</td>
<td>APACHE HADOOP (1219)</td>
<td></td>
</tr>
<tr>
<td>APACHE ANT (3927)</td>
<td>APACHE LOGGING (1131)</td>
<td>20 10 68 35</td>
</tr>
<tr>
<td>APACHE ANT (3927)</td>
<td>APACHE LUCENE (981)</td>
<td>24 12 61 34</td>
</tr>
<tr>
<td>APACHE ANT (3927)</td>
<td>APACHE MAVEN (1577)</td>
<td>74 37 261 143</td>
</tr>
<tr>
<td>APACHE HADOOP (1219)</td>
<td>APACHE LOGGING (1131)</td>
<td>2 1 19 10</td>
</tr>
<tr>
<td>APACHE HADOOP (1219)</td>
<td>APACHE LUCENE (981)</td>
<td>44 22 45 23</td>
</tr>
<tr>
<td>APACHE HADOOP (1219)</td>
<td>APACHE MAVEN (1577)</td>
<td>20 10 20 10</td>
</tr>
<tr>
<td>APACHE LOGGING (1131)</td>
<td>APACHE LUCENE (981)</td>
<td>4 2 16 8</td>
</tr>
<tr>
<td>APACHE LOGGING (1131)</td>
<td>APACHE MAVEN (1577)</td>
<td>32 16 98 52</td>
</tr>
<tr>
<td>APACHE LUCENE (981)</td>
<td>APACHE MAVEN (1577)</td>
<td>16 8 27 14</td>
</tr>
</tbody>
</table>

The figure shows that our approach effectively identified and matched more software developer IDs in contrast to existing approaches. The reason behind this is the clustering of different software developer names associated with an email address (e.g., “muhammad aftab iqbal” and “aftab iqbal” associated with “aftab.iqbal@deri.org” etc.) by parsing the email headers (i.e., From: “Name Surname” <email address>). Later, those software developer names are used to match other email addresses whose email header’s “name surname” part matches with the software developer name. Based on the results (cf. Figure 4.2), we can say that it is not common practice for a software developer to adopt multiple email addresses that are similar to each other (email-email similarity) or have similarities between his/her name and corresponding email alias (i.e., name-email similarity). We believe that combining existing approaches (i.e., name-email, email-email and name scheme similarity approaches) with our proposed approach may produce even better matching results, which could be taken into account as a next step for future work in this direction.
In this section, we discuss in detail our research questions (cf. Section 4.1.2) based on the results we achieved through interlinking different and multiple IDs of software developers within and across different software projects.

Q1: WHAT IS THE ADDED BENEFIT OF INTERLINKING MULTIPLE IDS OF SOFTWARE DEVELOPERS?

In order to evaluate the added benefits of interlinking multiple IDs of software developers within a software repository, we calculate the number of bugs reported by software developers who have used multiple IDs on a bug repository and that those IDs are not interlinked previously. In particular, we want to investigate how much of the software developer’s development activity is not explicitly interconnected due to the usage of multiple IDs within a software repository.

The multiple IDs of software developers were computed previously for a bug repository and results were shown in Table 4.7. For each Apache project, we consider only those software developers who have used multiple IDs in bug repository (cf. Table 4.7). For every software developer, we compute the total number of bugs reported by him/her using his/her multiple IDs by querying bug repository data sets. We plot the results for every Apache project in the form of stacked bar charts and are shown in Figure 4.3. We stacked up the bugs reported by a software developer based on his/her multiple IDs, in order to present the total number of bugs he/she reported.
Figure 4.3: Bugs reported by software developers using multiple IDs.

The results depicted in Figure 4.3 shows that establishing owl:sameAs links between multiple IDs of a software developer enables a complete view on the activity of a software developer within a particular software repository. Due to the selection of small number of Apache projects for this study, the bugs reporting activity using multiple IDs are not significant. However, we believe that we may find significant differences if it is applied to a large number of open source software projects.

Q2: What is the ratio of software developer’s existence in multiple software projects?

Based on the interlinking results achieved, we queried owl:sameAs links of each software developer for every Apache project under consideration. Later, we filtered only those software developers who have an owl:sameAs link pointing to other Apache projects and plotted the results as shown in Figure 4.4.
4.5 DISCUSSION

The chart shows that software developer’s participation in multiple software projects follow power’s law distribution because of the large number of software developers who appear to participate in at-least 2 Apache projects in contrast to 3, 4 or 5 Apache software projects. Moreover, for each Apache project we found a small subset of software developers who appear to participate in at-least 4 Apache projects. Thus, it shows a great tendency of software developer’s participation in multiple software projects. Moreover, it opens up new research challenges in order to assess, evaluate and analyze software developer’s development behavior while contributing to multiple software projects in a given period of time.

Q3: WHAT ARE THE CONTRIBUTIONS MADE BY SOFTWARE DEVELOPERS IN MULTIPLE SOFTWARE PROJECTS?

In order to trace software developers development activities across multiple software projects, we query and retrieve owl:sameAs links of each software developer. Later, we compute the development activities of only those software developers who have participated in multiple software projects. In Figure 4.5, we have summarized the development activities of some software developers who have contributed to or participated in at-least 2 Apache projects.

The figure shows that a small number of software developers have source control commit rights on multiple Apache projects while most software developers have commit rights on a single Apache project. Further, it shows that software developers are mostly active in exchanging emails and participating in bugs related discussions in contrast to reporting bugs while contributing to
multiple Apache projects. There is a possibility that software developers contributed to the code base of other Apache projects by submitting source code patches to the mailing list or bug tracking repository. We opt out this particular study as it requires extracting authorship information from source code patches or source control commit logs (e.g., “patch provided by John. Fixes LUCENE-1234 ...”) and further interlinking it to the respective software developers, which is not in the current scope of this work.

We believe that if the same methodology is applied to a large number of open source software projects, then we will see a significant amount of software developers development activities across multiple software projects.

4.6 Conclusion

In this chapter, we highlighted the problem of multiple IDs that are used by software developers while interacting with different software repositories of a software project. We outlined different types of IDs that software developers use in order to interact with software repositories and presented existing approaches to identify and relate different IDs of a software developer. Further, we applied existing approaches on our test dataset and found that they are not sufficient to address the problem of identifying multiple IDs of a software developer. Thus, we devised and presented our approach to identify and interlink multiple IDs of software developers. We applied our approach to different software repositories of a software project as well as across software repositories of
different software projects. In order to show the benefit of our proposed approach, we presented some questions and show that explicitly establishing the interconnection (i.e., Linked Data approach) between different IDs of software developers helps in integrating, querying and tracking software developers development activities in different software repositories of a software project as well as across multiple software projects.

While manually looking at a subset of multiple email addresses that belongs to software developers, we found cases where software developers contributed to a specific Apache project over the period of time using email addresses that are associated with different organisations/companies. This is most likely due to the fact that these software developers moved from one company to another over the period of time while still contributing to open source software projects. Based on this finding, we believe that it will be an interesting future work to investigate if these software developers contributed to Apache projects based on their intrinsic (fun, learning etc.) or extrinsic (financial rewards, career growth etc.) motivations or both [Krishnamurthy, 2006; Lakhani and Wolf, 2005]. For example, if the development activities of software developers are happened to be within daytime (office hours) then we can assume that these software developers are hired by companies specifically to contribute to these Apache projects (in order to meet some requirements of the company) because of their development experience on these software projects.

As a future work along the lines of software developer ID identification, one can combine existing software developer identification methods with our proposed methodology which may help in improving the interlinking approach, yielding higher quality and quantity links for software developer IDs. As a next step, we integrate other potential software project and software developer related open source repositories (e.g., statistical services, code forges etc.) in Chapter 5. Further, in Chapter 7 we study social dependencies and social relations among software developers over the period of time based on interconnected software developers IDs across different software repositories. We believe that it enables us to get useful insights into the social and technical aspects of software developers.

Just saw your technical paper on “Integrating FLOSS repositories on the Web”. Love the work, I think it would be hugely useful to get this done :) — James Howison, 2012

Managing activities within a software project using a variety of software repositories make the software development process look quite different than it used to be. Software developers are geographically distributed, they are agile and like to collaborate on different software projects with software developers across the world. Code forges are a good example of global software development [Herbsleb and Moitra, 2001]. It has gained a lot of attraction from the public and the software engineering community over the past decade. The success of these code forges is highly dependent on the infrastructure provided to the software developers and users in order to collaborate with each other [Shibuya and Tamai, 2009], which helps to increase transparency, feedback, trust and tracking of dispersed software developers. Code forges provide support for many software tools such as discussion forum, mailing list, bug tracking repository, source control repository, etc. Providing collaborative tools support [Sarma, 2005] in code forges promotes distributed software development. For example, a group of software developers are geographically distributed and interested to collaborate on a new software project. They are not required to setup the necessary Web-based software tools because hosting or creating a software project on a code forge automatically enables most of these software tools for the software developers in order to collaborate with each other on a software project.

Code forges host thousands of open source software projects that are driven by software developers who are geographically distributed. Often, software projects that are currently under development or have reached a mature state at different code forges share similarity in terms of functionality, programming language, database, operating system support etc. Moreover, software developers are often found contributing to software projects at different code forges and sometimes a software project migrate from one code forge to another over the period of time. Therefore, in this chapter we propose the integration of code forges based on metadata (e.g., programming language, database, intended audience, operating system etc.), similar software projects and software developers. Moreover, we present integration of software developer related information across different open source software repositories in order to show the benefits it brings to software stakeholders. The remainder of this chapter is structured as follows:

- In Section 5.1, we discuss the issues in integrating metadata of different code forges and propose to model metadata of different code forges using RDF vocabularies for integration purposes. We argue the benefits of interlinking code forges to each other (based on metadata) and to other relevant data sources on the Web through some example scenarios. Moreover, we compute the overlapping between code forges based on similar software developers and further, support the benefit of interlinking similar software developers and software projects across different code forges through couple of use cases.
In Section 5.2, we discuss in detail software developer related information that are extracted from different open source software repositories (i.e., code forges) and further interlink it to other relevant data sources on the Web (i.e., statistical services such as OML). We present some exemplary SPARQL queries that are executed on the integrated information in order to show that we are able to not only query software developers participation in multiple software projects across different code forges but also statistical information about a software project or software developer.

In Section 5.3, we conclude our work.

5.1 INTEGRATING CODE FORGES

With the success and adoption of open source software development, we have seen a tremendous growth in the availability and usage of different code forges [Squire and Williams, 2012]. Code forges provide different kinds of features in order to keep existing software projects and attract more software projects. These features include managing software project code, a place for the users to download the software project releases, discussion forum, bug tracking repository, mailing list etc. A software project team may choose to host the software project on their own code repository or may choose to host the software project on one of the many available online code forges (e.g., SourceForge, Savannah¹, GitHub etc.). Each software project code repository along with other tools leave a detailed trace about the activities being carried out during the whole life span of a software project [Gonzalez-Barahona et al., 2010]. As pointed out by [Conklin, 2006], it is still surprisingly difficult to obtain and abstract information from these software repositories in order to answer even simple questions like:

- how many software developers are working on a software project?
- how big the community is surrounding a software project?
- how many contributors are contributing to a software project?
- what is the development ratio per software developer?
- is the software project flourishing?

These are some questions that are hidden deep inside the software repositories and usually have in the minds of software developers before joining or start contributing to a software project. Having answers to such questions can give a clear picture to the newcomers or other interested users/software developers of a software project. Much software development research has been carried out on gathering metrics and developing empirical studies based on the data retrieved from these software project repositories. However, researchers sometime finds it difficult to make sense of all the data for a research study due to the sheer size of code forges, the amount of data each software project holds, the heterogeneity of the software projects being studied and harvesting or crawling the code forges, which is often a daunting task. In order to provide researchers an easy access to the software project’s data, two research software projects are initiated (with slightly different objective) by the FLOSS research community, which are FLOSSMole² [Howison et al.,

¹ http://savannah.gnu.org/
² http://flossmole.org/
and FLOSSMETRICS\textsuperscript{3} [Herraiz et al., 2009] and also known as “repository of repositories (RoR)”. These RoRs are created to consolidate metadata (FLOSSMOLE) and analysis of software projects (FLOSSMETRICS) from a variety of code forges into a centralized place for use by the researchers in academia and industry.

Our main focus in this section is to demonstrate integration of code forges based on metadata, similar software projects and software developers. Therefore, we take into consideration software project’s metadata for different code forges that are made available to download by the FLOSSMOLE community rather than crawling our own data. FLOSSMOLE crawls data from a variety of code forges but we consider only GoogleCODE, SOURCEFORGE and GitHUB as a case study in this section, although our methods extend to other code forges as well.

### 5.1.1 Motivation

FLOSSMOLE [Howison et al., 2006] crawls only meta information of software projects (such as, software project name, description, url, no. of software developers, programming language, operating system, license etc.) and software developers (such as, software developer name, software developer ID, software projects he/she is contributing to etc.) from different code forges and makes each code forge metadata accessible to the community as flat delimited files, SQL dumps or direct database access respectively. Although the data extracted from each code forge is complete and well described, integrating knowledge across different code forges is still a big challenge as we have argued in [Iqbal et al., 2012a].

Each code forge has its own database schema and represents data elements differently. Code forges use different terminologies or keywords that allows development teams to select and associate metadata from various categories (e.g., software project topics, operating systems, programming languages etc.) to a particular software project. For example, GoogleCODE defines the term “Mac” to associate MACINTOSH as an operating system to a software project but SOURCEFORGE defines it as “OS X”. Although both refers to the same operating system, different terms are used. Hence, we require methods and techniques to define some kind of classification/mapping which explicitly states that the two terms are semantically similar and belong to the same operating system family. By doing so, we are able to run a query on different code forges in order to retrieve a list of software projects that supports MACINTOSH operating system.

Associating metadata to a software project is also being handled differently in different code forges. For example, SOURCEFORGE provides a granular hierarchical structure for associating metadata to a software project by allowing software developers to select metadata from various category lists (e.g., operating systems, programming languages, database environments etc.). Comparing it to GoogleCODE and GitHUB, there is no proper hierarchical structure for associating metadata to a software project, which makes it harder to integrate code forges based on similar metadata. Furthermore, code forges do not share common semantics for associating metadata to a software project at the database schema level. For example, we cannot say that the database table column named description in SOURCEFORGE holds the same meaning as the database table column named label in GoogleCODE. The reason is column label hold values from various categories (e.g., programming languages, software project topics, databases, etc.).

Further challenges upfront (as mentioned by [Conklin, 2006]) while integrating code forges are:

1. **code forges have software projects with similar names**: Large open source software projects often have sub-software projects hosted at different code forges or sometimes software projects

\textsuperscript{3} http://www.flossmetrics.org/
are hosted at multiple code forges exploiting different development infrastructures. For example, Apache Software Foundation provides its own infrastructure to host Apache software projects but it also provide mirrors of various Apache software projects on GitHub to those software developers who prefer git versioning system\(^4\) over svn versioning system\(^5\). (2) **identifying software developers across different code forges**: Software developers are often found on multiple code forges due to the existence of similar software projects on these code forges. However, identifying similar software developers across different code forges is not simple because:

- software developers may use different IDs for different code forges;
- different software developers may have similar names/IDs across different code forges;
- software developers may use similar IDs across different code forges.

Apart from the challenges upfront while integrating information about software developers and software projects across different code forges, it is not wrong to state that code forges are somehow interconnected but rather of an implicit nature – for example, software developers contributing to different software projects on multiple code forges. Hence, we need to make the interconnections among code forges explicit and further allow connecting it to other relevant data on the Web. Having an explicit representation of the interconnections between different code forges, we are able to support certain use case scenarios, some of which are listed in the following:

- **Enabling complete development history of software projects distributed on multiple code forges**. Sometimes software projects migrate from one code forge to another over the period of time. Linking similar software projects enable full development history of software projects across different code forges.

- **Tracing software developers activities across different code forges**. Linking software developers profile across different code forges enable us to discover the activities of software developers across different code forges, a variant of which is described already in Section 4.5. It further allows us to rank code forges based on the number of software projects they are working on in a particular code forge.

- **Is the popularity level of a particular code forge increasing or decreasing?** Assuming that code forges are connected to each other, it is interesting to investigate if software developers are migrating from one code forge to the other. For example, if software developers are considering hosting new software projects on GitHub rather than GoogleCode or SourceForge. This use case lay down the foundations to study the dynamics of open source software community across different code forges.

- **Assuming that software project’s metadata is interlinked to other data sources on the Web**, further information about a particular entity/artifact can be provided. For example, if Algol 68\(^6\) is associated as a programming language to a particular software project than the software developer is able to retrieve further information about Algol 68 from the Web (for example, from Wikipedia\(^7\)).

---

\(^4\) [http://git-scm.com/](http://git-scm.com/)


\(^6\) [http://www.algol68.org/](http://www.algol68.org/)

In order to deal with interoperability and integration issues to realize different use case scenarios, we explain modeling of code forges metadata based on a standard format in the following subsection.

5.1.2 Modeling Code Forge’s Metadata

As mentioned earlier, the usage of a common model and standard format is required to represent software project’s metadata from different code forges for better integration. One may think of questions like: what is the best way to express knowledge so that it can be integrated easily across multiple code forges? Can the knowledge be further used to link to other data sources that contains extra information about a certain entity? Can it be done in an automated fashion? How easy will it be to explore related knowledge from different data sources?

As discussed in detail in Section 3.1, RDF provides a flexible way to represent information of any real-world domain in the form of statements about resources and how are they related to each other. Therefore, we propose Linked Data approach to represent software project’s information (i.e., metadata) contained inside different code forges. We looked into SourceForge, GoogleCode, and GrtHub’s hierarchical structure in order to identify the process of associating metadata to a software project and found that SourceForge is following an organized hierarchical structure of categorizing and associating metadata to software projects. Therefore, we do the modeling based on the hierarchical organization of metadata in SourceForge rather than GoogleCode or GrtHub.

Each code forge provides a list of keywords/terminologies for various categories (e.g., programming language, operating system etc.) that can be used to associate metadata to a software project. Therefore, we use SKOS vocabulary for modeling because SKOS vocabulary allows to aggregate concepts/terminologies into a single concept scheme. An excerpt of an exemplary RDF representation of an operating system category using SKOS vocabulary is shown in Listing 5.1

```
@prefix base: <http://example.org/schemes/os#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
base:modern a skos:ConceptScheme;
skos:prefLabel "Modern (Vendor-Supported) Desktop Operating Systems"@en;
skos:hasTopConcept base:linux;
skos:hasTopConcept base:net-bsd;
skos:hasTopConcept base:os-x

base:linux a skos:Concept;
skos:inScheme base:modern;
skos:prefLabel "Linux"@en
.
.
```

Listing 5.1: An exemplary SKOS concept scheme that defines different operating systems for SourceForge.

The benefit of using SKOS vocabulary is that the concept schemes are easily extensible. Therefore if a code forge later adds a new operating system, which is not listed in the existing concept

---

8 We encourage readers to have a look at the concept schemes, which are available at: http://srvgal85.deri.ie/linkedfloss/schemes/
scheme than it can be added easily (see line #3 and lines #5–7 of Listing 5.2). Moreover, we can define SKOS concept schemes for each code forge and later link it to the core SKOS concept scheme using skos:inScheme property. This way we keep all predefined operating systems for different code forges under one scheme. As mentioned earlier (cf. Section 5.1.1) about the usage of different terminologies by code forges for the same concept, SKOS vocabulary offers to use skos:exactMatch property in order to express that the terms (e.g., “Mac” and “OS X”) are semantically similar.

```plaintext
1 base:modern a skos:ConceptScheme;
2 ...
3 skos:hasTopConcept base:vista;
4 ...
5 base:vista a skos:Concept;
6 skos:inScheme base:modern;
7 skos:prefLabel "Windows Vista"@en
8 ...
9 ...
```

Listing 5.2: Adding a new concept to the existing SKOS concept scheme.

Once code forges are modeled based on metadata (i.e., attributes), the next step is to interlink the metadata to other relevant data sources on the Web. Referring back to Listing 5.2, we know that “Windows Vista” is an operating system and it is beneficial if we interlink this concept to other relevant data source on the Web (for example, to the WIKIPEDIA entry9). By interlinking the two data sources together, we are able to retrieve information about “Windows Vista” from WIKIPEDIA. In order to extract structured information from WIKIPEDIA and represent it as RDF statements, the research community has developed DBPEDIA [Auer et al., 2007]. Therefore, we interlink the concepts defined in our concept scheme (e.g., base:vista in this example) to the corresponding DBPEDIA entity/resource using an owl:sameAs property indicating that these URIs actually refer to the same entity (see Listing 5.3). A partial list of SKOS concept scheme for operating systems that are defined for SOURCEFORGE can be found in Listing C.1.

```plaintext
1 ...
2 base:vista a skos:Concept;
3 skos:inScheme base:modern;
4 skos:prefLabel "Windows Vista"@en;
5 owl:sameAs <http://dbpedia.org/resource/Windows_Vista>
6 ...
7 ...
```

Listing 5.3: An interlinking example connecting a metadata term to its relevant entity in DBPEDIA.

We have shown how to model metadata of a code forge using SKOS vocabulary and further provide an example of interlinking it to other relevant data sources on the Web, the next step is to use this meta information while publishing RDF descriptions of software projects that are contained inside FLOSSMOLE database dumps. In order to represent a software project (contained inside a code forge) as RDF statements, we follow the same principles that have explained earlier in Section 3.2.3. Due to the large number of software projects contained inside FLOSSMOLE database dumps, we have developed custom written scripts to automatically publish each software project as RDF statements based on the information provided by FLOSSMOLE. We show RDF representation of 1 software project taken from the SOURCEFORGE database dump, as an example

in Listing 5.4. We use different RDF vocabularies to describe meta information of software projects as RDF statements. *doap* (description of a project) vocabulary is specifically designed to describe meta information of a software project, therefore we use various properties from *doap* vocabulary to describe information of a software project as RDF statements. Moreover, we use *foaf* vocabulary to describe meta information of a software developer as shown in Listing 5.4.

```xml
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

http://example.org/sourceforge/project/filezilla a doap:Project;
  doap:name "FileZilla";
  doap:created "2001-02-27T00:00:00.000Z";
  doap:description "FileZilla is a cross-platform graphical FTP, FTPS and SFTP client with a lot of features, supporting Windows, Linux, Mac OS X and more. FileZilla Server is a reliable FTP server for Windows.";
  doap:developer <http://example.org/sourceforge/author/botg>,
               <http://example.org/sourceforge/author/ci-dev>,
               <http://example.org/sourceforge/author/eddan>;
  doap:homepage <http://sourceforge.net/projects/filezilla>;
  doap:license <http://www.gnu.org/licenses/gpl-1.0.html>;
  foaf:homepage <http://filezilla.sourceforge.net>;
  dct:subject <http://example.org/schemes/audience#desktop>,
               <http://example.org/schemes/audience#developers>,
               <http://example.org/schemes/topic#social_networking>,
               <http://example.org/schemes/translation#french>,
               <http://example.org/schemes/translation#english> .

<http://example.org/sourceforge/author/botg> a foaf:Person;
  foaf:accountName "botg";
  foaf:mbox_sha1sum "9080a9d64e29e35892c9517bd95e2e8e4f5b13f6";
  foaf:name "Tim Kosse" .
```

Listing 5.4: RDF representation of a SOURCForge software project based on the data provided by FLOSSMole.

As each code forge has its own database schema, the RDF representation of software projects slightly differs for each code forge. An exemplary RDF representation of GtHUB and GOLGCODE software projects are also listed in Listing C.2 and Listing C.3. The overall concept of integrating code forges is also depicted in Figure 5.1, which basically covers the steps as described in the following:

1. The software project’s metadata from different code forges is being crawled periodically by FLOSSMole and FLOSSMETRICS, yielding database dumps of each code forge.
2. Study database schema of each code forge and model it using a common vocabulary, such as SKOS.
3. Generate RDF description of each software project found in code forges.
4. Interlink metadata of software projects to each other, across code forges and to the LOD cloud\(^\text{10}\), where necessary.

\(^\text{10}\)[http://lod-cloud.net/](http://lod-cloud.net/)
5.1.3 Interlinking Code Forges: Preliminary Findings

In this section, we argue the benefit of interlinking code forges to each other and to other potential data sources on the Web through examples. In Table 5.1, we show the total number of software projects and software developers that exist in the FLOSSMole database dumps of different code forges.

<table>
<thead>
<tr>
<th>Code Forge</th>
<th>Software Projects</th>
<th>Software Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceForge</td>
<td>204439</td>
<td>256516</td>
</tr>
<tr>
<td>GoogleCode</td>
<td>167286</td>
<td>203691</td>
</tr>
<tr>
<td>GitHub</td>
<td>155326</td>
<td>119196</td>
</tr>
</tbody>
</table>

Table 5.1: Total number of software projects and software developers found in the FLOSSMole database dumps of different code forges.

Once the software projects and software developers of each code forge under consideration are published as RDF datasets using our custom written scripts, we loaded the RDF datasets into a SPARQL endpoint. We discussed earlier about interlinking code forges metadata to their relevant terms in DBPEDIA. In Table 5.2, we show the total number of owl:sameAs links established between DBPEDIA and relevant code forges metadata terms that are defined using SKOS vocabulary.
Table 5.2: Interlinking code forge metadata terms with DBpedia dataset.

The owl:sameAs links are created by manually checking the terms in a code forge and their corresponding entry in DBpedia. However, duplicate detection algorithms and framework like Silk [Volz et al., 2009], Swoosh [Benjelloun et al., 2009], Duke etc., can also be taken into account for establishing the owl:sameAs links but it is not in the current scope of this work. The results show that SourceForge has defined a wide variety of keywords/attributes to associate metadata to a software project in contrast to Googlecode and GitHub.

In this section, we considered and performed interlinking of code forges metadata terms with DBpedia dataset. However, other potential LOD datasets can also be taken into account to enrich the interlinking between code forges metadata and other LOD cloud datasets. This is particularly beneficial in querying relevant information about a particular entity from multiple data sources. For example, a software project is written in Java programming language and we link Java with DBpedia resource using owl:sameAs property, which allows us to query DBpedia to find out more information about Java programming language. For example:

- who is the designer/software developer?
- when was it created?
- what is the license type?
- what operating system does it support?

Another benefit of interlinking is the ease of querying software projects across code forges that comes under a specific category. For example, one query could be to find out the total number of software projects hosted on different code forges that are implemented in a specific programming language. We consider 5 programming languages and show the number of software projects that are implemented using these programming languages on different code forges, in Table 5.3. The results shown in Table 5.3 can be further narrowed down by specifying other selection criteria in the query (e.g., audience, database, operating system etc.).

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Googlecode (software projects)</th>
<th>GitHub (software projects)</th>
<th>SourceForge (software projects)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>19,760</td>
<td>22,331</td>
<td>8,975</td>
</tr>
<tr>
<td>Ruby</td>
<td>2,163</td>
<td>28,655</td>
<td>1,563</td>
</tr>
<tr>
<td>Delphi</td>
<td>418</td>
<td>135</td>
<td>3,023</td>
</tr>
<tr>
<td>MatLab</td>
<td>393</td>
<td>181</td>
<td>441</td>
</tr>
<tr>
<td>Assembly</td>
<td>162</td>
<td>196</td>
<td>2,058</td>
</tr>
</tbody>
</table>

Table 5.3: Number of software projects developed using different programming languages on code forges.
We now look into two different examples from the point of integrating code forges to each other based on the existence of similar software developers and integrating software projects to their relevant information contained in other data sources available on the Web.

**Example 1: Interlinking Software Projects to Other Relevant Data Resources.**

We provide an example of interlinking a software project hosted on a code forge to its existence in other data sources on the Web. **O**HLOH is a free wiki for open source software projects and developers. **O**HLOH provides statistical information and services about open source software projects by crawling data from a variety of code forges. There also exists an RDF wrapper, a.k.a., RDF**O**HLOH [Fernández, 2008], which provides Linked Data version of **O**HLOH. Therefore, RDF description of a software project generated by RDF**O**HLOH can be interlinked to its relevant software project’s RDF description generated from the code forge using an owl:sameAs property. Such an interlinking allows anyone to query statistical and other relevant information about a software project or software developer from **O**HLOH as shown in the example below.

We take the example of FileZilla software project, which is hosted on **S**ource**F**orge and its statistical information is available at **O**HLOH. Given that we have the metadata of FileZilla software project available in RDF (cf. **L**isting 5.4) and the RDF description of FileZilla software project from **O**HLOH is made available using RDF**O**HLOH (see **L**isting C.4), we interlink both RDF datasets based on software developer IDs. Once the RDF datasets are interlinked, we are able to query statistical information about a software project, geo location of a software developer, total number of commits made by a software developer, software developer’s kudo rank and other software projects he/she is contributing to etc., from **O**HLOH. An exemplary SPARQL query that retrieves information from **O**HLOH about a software developer working on FileZilla software project is show in **L**isting 5.5.

```sparql
PREFIX doap: <http://usefulinc.com/ns/doap#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX lf: <http://vocab.deri.ie/linkedfloss#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

select ?name ?kudorank ?lat ?long {
?prj doap:name "FileZilla" .
?prj doap:developer ?d .
?d owl:sameAs <http://rdfohloh.wikier.org/user/19797#person> .
?dev foaf:name ?name .
?f foaf:holdsAccount ?dev .
?geolocation geo:lat ?lat .
}
```

Listing 5.5: A SPARQL query that retrieves information about a software developer working on FileZilla project from **O**HLOH.

The results returned by the query listed in **L**isting 5.5 is shown in **T**able 5.4. Through interlinking a software project hosted on a code forge with **O**HLOH, we are able to track not only software
developer’s development activity on a particular software project by mining information hidden deep inside the software repositories but also statistical information about software developer’s development activity.

Table 5.4: Geo location and rank of a software developer retrieved from OHLOH.

<table>
<thead>
<tr>
<th>name</th>
<th>kudorank</th>
<th>lat</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tim Kosse</td>
<td>10</td>
<td>51.165691</td>
<td>10.451526</td>
</tr>
</tbody>
</table>

EXAMPLE 2: SOFTWARE DEVELOPERS OVERLAPPING ACROSS CODE FORGES.

We now perform a preliminary study to identify the overlap between different code forges based on the existence of software developers on multiple code forges. For this preliminary study, we also extracted 29,860 distinct <name,email address> pairs of people (i.e., software developers, contributors, bug reporters, users etc.) from the bug repository of Apache Software Foundation\(^{15}\) for 27 randomly selected Apache software projects. We strictly matched their <name,email address> (excluding the domain after “@”) pairs against all software developers <name,email address> (excluding the domain after “@”) pairs that are found in the database dump of SourceForge. The database dump of Googlecode provided by FLOSSMole do not contain naming information about software developers, therefore we matched only software developer IDs in the case of Googlecode. The results of this preliminary study is shown in Table 5.5. The analysis may contain errors because there may be different software developers with identical <name,email address> (excluding the domain after “@”) pairs exist in different code forges. Furthermore, we have less confidence on our matching results against Googlecode because we only matched software developer IDs. A more thorough validation is required before interlinking software developers across code forges. However, our preliminary study shows that there are overlapping across different code forges and these overlapping can be made explicit by interlinking software developers and software projects, hence developing an interlinked FLOSS ecosystem.

<table>
<thead>
<tr>
<th>Forge A (software developers)</th>
<th>Forge B (software developers)</th>
<th>Software Developers Found</th>
<th>Overlap Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache (29,860)</td>
<td>SourceForge (256,516)</td>
<td>1,480</td>
<td>4.95</td>
</tr>
<tr>
<td>Apache (29,860)</td>
<td>Googlecode (203,691)</td>
<td>1,041</td>
<td>3.48</td>
</tr>
<tr>
<td>Googlecode (203,691)</td>
<td>SourceForge (256,516)</td>
<td>16,351</td>
<td>8.02</td>
</tr>
</tbody>
</table>

Table 5.5: Overlapping between code forges based on the existence of software developers on multiple code forges.

For our study, we considered only those people who are involved in communication on the bug repository of 27 different Apache software projects. Most of them may not be the actual developers of those Apache software projects. However, we wanted to show that the software developers who exists on SourceForge or Googlecode are somehow contributing (e.g., reporting bugs, commenting on bugs, submitting source code patches, etc.) or part of Apache software projects too and their contributions can be made explicit by interlinking them.

\(^{15}\) [https://issues.apache.org/bugzilla/](https://issues.apache.org/bugzilla/)
5.1.4 Discussion

In the previous subsection, we discussed briefly the implicit connection between code forges based on software developers existence in multiple code forges. Further, we showed the advantage of interlinking software projects to other relevant data sources (e.g., OAh) on the Web, hence allowing to retrieve more information relevant to a software project or software developer. In the following, we further support the idea of interlinking software projects and software developers across different code forges by presenting 2 different scenarios. We emphasize that we do not propose any heuristics or algorithm for efficient software project and software developer matching here but instead advocate the benefits of interlinking code forges through these scenarios.

**Scenario 1: Interlinking similar software projects across different code forges**

In order to identify the existence of similar software projects across code forges, different heuristics can be applied as proposed by [Conklin, 2007]. For example, software project’s metadata (e.g., name, description, URL, license, operating system, programming language etc.) can be taken into account to match software projects across code forges. However, we just want to show that once each software project is assigned a URI and further described as RDF statements using different RDF vocabularies, we are able to integrate similar software projects across different code forges by establishing links between software project URLs. Among different heuristics that are proposed by [Conklin, 2007], we take into account only software project URL as a matching attribute in order to identify similar software projects across code forges.

Software projects hosted on GOOGLECODE have URLs of the form: http://code.google.com/p/x/, where x is the name of a software project. We query all software projects hosted on SOURCEFORGE that contains GOOGLECODE software project URLs. For all those software projects where GOOGLECODE software project URLs are found, we establish an owl:sameAs link between the URIs of both software projects. For an example, we consider QUADRA software project, which is a multiplayer action puzzle game and is hosted on SOURCEFORGE and GOOGLECODE. Assuming that the software projects of SOURCEFORGE and GOOGLECODE are available as RDF datasets, we link both software projects using an owl:sameAs property as shown in Listing 5.6.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
<http://example.org/googlecode/project/quadra> owl:sameAs <http://example.org/sourceforge/project/quadra> .
```

Listing 5.6: An interlinking example connecting similar software project on GOOGLECODE and SOURCEFORGE.

Matching software projects hosted on SOURCEFORGE against software projects hosted in other code forges is achieved in a slightly different way. SOURCEFORGE usually have two different URLs for a single software project. One URL refers to the actual software project directory where software repositories, software project’s metadata and software developers information are available (e.g., http://sourceforge.net/projects/x), while the other URL refers to the HTML website that describes the software project (e.g., http://x.sourceforge.net/). The software development teams may use any of the two URLs as a homepage for the software projects that are hosted

---

16 https://code.google.com/p/quadra/
17 http://sourceforge.net/projects/quadra/
on other code forges. Therefore, we match software projects based on both URLs. We apply these approaches in a pairwise manner on Googlecode, GitHub and SourceForge code forges. The results achieved through matching software projects based on URL across different code forges are shown in Table 5.6.

<table>
<thead>
<tr>
<th>Source Code Forge</th>
<th>Target Code Forge</th>
<th>Software Projects Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>Googlecode</td>
<td>SourceForge</td>
<td>177</td>
</tr>
<tr>
<td>Googlecode</td>
<td>GitHub</td>
<td>595</td>
</tr>
<tr>
<td>SourceForge</td>
<td>GitHub</td>
<td>699</td>
</tr>
<tr>
<td>GitHub</td>
<td>SourceForge</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 5.6: Matching similar software projects across different code forges based on their URLs.

The results in Table 5.6 shows existence of good number of Googlecode and SourceForge software projects that are found on GitHub in contrast to GitHub software projects that are found on SourceForge. This can be due to the increase in the preference of software developers working in a GitHub environment or an indication of the growing popularity of GitHub over Googlecode and SourceForge. The Googlecode database dump (provided by Flossmo) do not have any information about software project URLs, which is the reason we do not provide any results of the existence of SourceForge and GitHub software projects that may exist on Googlecode. The number of matched software projects based on the software project URLs is relatively low comparing to the number of software projects hosted on each code forge (cf. Table 5.1). However, one advantage of matching software projects based on the URL is that it also matches those software projects where the software project names are different across code forges. For example, libxls\(^{18}\) and libxls\(_{\text{cmake}}\)^{19} are same software projects hosted on different code forges where libxls\(_{\text{cmake}}\) contains software project URL of libxls. We may achieve a high number of matching candidates if we take into account other matching attributes that are mentioned by [Conklin, 2007]. However, our goal here is not to increase the number of matched software projects across code forges but to show the benefits of interlinking and exploiting software project-related information across different code forges.

Given that similar software projects are interlinked across different code forges through owl:sameAs property, we are able to extract a list of software developers who are working on the same software project across different code forges as shown in Listing 5.7.

```sparql
PREFIX doap: <http://usefulinc.com/ns/doap#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
select ?developers {
  ?prj a doap:Project .
  ?prj doap:name "quadra" .
  {?prj owl:sameAs ?prjAlt .} UNION {?prjAlt owl:sameAs ?prj .}
}
```

Listing 5.7: A SPARQL query that retrieves a list of software developers working on the same software project (i.e., quadra) across different code forges.

---

\(^{18}\) http://sourceforge.net/projects/libxls/

\(^{19}\) https://github.com/wiglot/libxls_cmake
The results of the query (cf. Listing 5.7) are shown in Table 5.7. The results in Table 5.7 shows that 4 software developers are found on GoogleCode developing the “quadra” software project while other software developers are found on SourceForge.

<table>
<thead>
<tr>
<th>Software Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/googlecode/author/pphaneuf">http://example.org/googlecode/author/pphaneuf</a></td>
</tr>
<tr>
<td><a href="http://example.org/googlecode/author/rveilleux">http://example.org/googlecode/author/rveilleux</a></td>
</tr>
<tr>
<td><a href="http://example.org/googlecode/author/slajoie">http://example.org/googlecode/author/slajoie</a></td>
</tr>
<tr>
<td><a href="http://example.org/googlecode/author/dgryski">http://example.org/googlecode/author/dgryski</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/pphaneuf">http://example.org/sourceforge/author/pphaneuf</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/rveilleux">http://example.org/sourceforge/author/rveilleux</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/stanb">http://example.org/sourceforge/author/stanb</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/nypon">http://example.org/sourceforge/author/nypon</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/pihvi">http://example.org/sourceforge/author/pihvi</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/slajoie">http://example.org/sourceforge/author/slajoie</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/plombe">http://example.org/sourceforge/author/plombe</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/author/roncli">http://example.org/sourceforge/author/roncli</a></td>
</tr>
</tbody>
</table>

Table 5.7: List of software developers working on the same software project across different code forges.

An interesting fact we see in Table 5.7 is the existence of similar software developers in multiple code forges due to the existence of similar software projects. We see that some software developers (i.e., pphaneuf, rveilleux and slajoie) in Table 5.7 holds account IDs for both code forges (i.e., GoogleCode and SourceForge). Existence of software developers in multiple code forges can be exploited by interlinking their IDs (using Linked Data principles), which is discussed in the next scenario.

Scenario 2: Tracing Software Developer Activities Across Different Code Forges.

As observed in Table 5.7, the existence of software projects in multiple code forges increases the chance of software developers existence in these code forges. Therefore, identifying and interlinking software developer IDs across code forges allow us to keep track of software developers involvement in different software projects across code forges. Moreover, expertise profile of software developers can be constructed by mining software development activities of these software developers on different software projects across code forges.

In order to identify the existence of software developers in multiple code forges, we consider software developers of only those software projects that are found on multiple code forges (based on the results of Table 5.6). We query a list of similar software projects that exist on different code forges by exploiting owl:sameAs links. For each software project found on multiple code forges, we extract a list of software developers who are working on it in different code forges. Later, we match the software developers by simple string similarity algorithm. If a match is found, we establish an owl:sameAs link between the two software developer IDs as shown in Listing 5.8. The total number of matched software developers based on this simple matching approach is shown in Table 5.8.

We may achieve a high number of matched software developers if we consider all software developers of each code forge but our focus in this chapter is only to demonstrate the benefit of identifying and interlinking similar software projects and software developers across different
Listing 5.8: An interlinking example connecting IDs of the same software developer found in GoogleCode and SourceForge.

<table>
<thead>
<tr>
<th>Forge A (software developers)</th>
<th>Forge B (software developers)</th>
<th>Software Projects</th>
<th>Software Developers Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoogleCode (470)</td>
<td>SourceForge (243)</td>
<td>177</td>
<td>47</td>
</tr>
<tr>
<td>GoogleCode (2760)</td>
<td>GitHub (1590)</td>
<td>595</td>
<td>49</td>
</tr>
<tr>
<td>SourceForge (2962)</td>
<td>GitHub (862)</td>
<td>699</td>
<td>105</td>
</tr>
</tbody>
</table>

Table 5.8: Matching software developers based on similar software projects across different code forges.

code forges. Given that IDs of a software developer is interlinked across different code forges through owl:sameAs property, we are able to query different software projects on which a software developer is working, as shown in Listing 5.9.

Listing 5.9: A SPARQL query that retrieves all software projects of a software developer across different code forges.

The result of the SPARQL query (cf. Listing 5.9) is shown in Table 5.9. The result shows that “pphaneuf” is a developer of 5 software projects on GoogleCode and 4 software projects on SourceForge. With this preliminary information in hand, we can extract development activity of “pphaneuf” on these software projects by mining the information contained inside different software repositories as we have already demonstrated in Section 4.5.

In this section, we have motivated and proposed integration of code forges based on their meta-data, similar software projects and software developers. We highlighted the issues in integrating metadata of different code forges and proposed to use a common model to integrate code forges based on metadata. Further, we presented 2 different scenarios in order to argue the benefit of interlinking code forges to each other by integrating similar software projects. In the next section, we take into consideration different open source software repositories (code forge, Apache project, Ohloh) and discuss the benefits of interlinking them based on similar software developer profiles through some exemplary SPARQL queries.
5.2 INTEGRATING DEVELOPER INFORMATION ACROSS OPEN SOURCE REPOSITORIES

In the previous section, we looked into existing code forges data that are made available by FLOSSMOLE community. Due to the limited amount of information provided by FLOSSMOLE, we were not able to demonstrate queries that retrieve information hidden inside software repositories. Therefore, we overcome this limitation by extracting information from open source repositories (i.e., GitHub and Ohloh) and interlink them.

With the success and adoption of open source software development, we have seen a tremendous growth in the availability of different code forges. Different code forges provide different kinds of features in order to keep existing software projects and attract more software projects. Because of this, an open source software project sometimes developed at multiple code forges. For instance, Apache Software Foundation (ASF) manages software projects at their own infrastructure but also provide GitHub mirrors for software developers who prefer Git over SVN versioning system. Therefore, certain software developers use the GitHub infrastructure to contribute to the development of Apache software projects. At the time of writing this thesis, ASF host mirrors of approximately 552 different Apache software projects on GitHub. Furthermore, an open source software project sometimes migrate between different code forges during its entire time period.

Code forges require software projects and software developers to adopt an identity in order to host a software project and keep track of software developers development activities. Eventually, software developers end up developing multiple software projects in different code forges. For instance, Stefan Bodewig\(^{21}\) who is a member of ASF and is a contributor to multiple Apache software projects\(^{22}\), also found to be a developer of software projects hosted on GitHub\(^{23}\), Google-code\(^{24}\) and SourceForge\(^{25}\). Therefore, we can say that the history of open source software project development and software developer’s contribution to different open source software projects are distributed across different code forges.

---

Table 5.9: List of software projects on which a software developer is working across different code forges.

<table>
<thead>
<tr>
<th>Software Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/googlecode/project/google-glog">http://example.org/googlecode/project/google-glog</a></td>
</tr>
<tr>
<td><a href="http://example.org/googlecode/project/quadra">http://example.org/googlecode/project/quadra</a></td>
</tr>
<tr>
<td><a href="http://example.org/googlecode/project/quagmire">http://example.org/googlecode/project/quagmire</a></td>
</tr>
<tr>
<td><a href="http://example.org/googlecode/project/wvstreams">http://example.org/googlecode/project/wvstreams</a></td>
</tr>
<tr>
<td><a href="http://example.org/googlecode/project/xplc">http://example.org/googlecode/project/xplc</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/project/opencom">http://example.org/sourceforge/project/opencom</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/project/pasta">http://example.org/sourceforge/project/pasta</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/project/quadra">http://example.org/sourceforge/project/quadra</a></td>
</tr>
<tr>
<td><a href="http://example.org/sourceforge/project/xplc">http://example.org/sourceforge/project/xplc</a></td>
</tr>
</tbody>
</table>

---

20 https://github.com/apache
21 http://stefan.samaflost.de/
22 http://people.apache.org/committer-index.html#bodewig
23 https://github.com/bodewig
24 http://code.google.com/u/114630878996357081254/
25 http://sourceforge.net/users/bodewig
Moreover, there also exist statistical services that provides detailed analysis on open source software projects. One good example of such a service is Ohloh. Ohloh is a free, public software directory that monitors up-to-date development activity of open source software projects. Ohloh allows software developers to join (i.e., adopt an identity) and claim their contributions (i.e., commits) on existing software projects and also allow them to add software projects that are not yet on Ohloh, in order to assemble a complete profile of their open source software project contributions. Ohloh provides several types of information about an open source software project. For example, it provides information about each software developer’s commit ratio to a software project, each programming language commit ratio to a software project, longevity of a software project and software metrics such as total lines of source code, commit statistics, comment ratio etc. Other global statistics like programming-language usage are also provided. Ohloh provides all types of information that a software stakeholder is interested or keen to know. At the time of writing this thesis, Ohloh indexed 665,413 open source software projects connecting more than 3,497,433 open source software developers/contributors making it a valuable data source for collecting up-to-date metrics of open source software projects.

It is worth mentioning that there is an implicit connection between the software developer’s development activity in different software repositories (i.e., mailing lists, bug tracking systems, source control etc.) hosting a particular software project, software developer’s development statistics (available via Ohloh) and involvement in multiple software projects on different code forges. We need hence not only make the interconnection between software developer IDs among different software repositories within a software project explicit but also allow connecting it to other related data sources available on the Web as we have argued in [Iqbal and Hausenblas, 2012]. Having such an explicit representation of the interconnection between the data sources available, we are able to support certain scenarios often found in the software development process:

1. **Synthesis Scenarios**
   - A software developer could effectively query the development activities of co-workers in different software repositories of a software project.
   - A software developer could learn about the expertise of co-workers in different programming languages.
   - A software developer could easily track the contribution of co-workers in different software projects.

2. **Analysis Scenarios**
   - Different programming languages used in a software project and the ratio of commits made through each programming language.
   - Development ratio of a software project across multiple code forges.
   - Software developer’s development statistics on each software project.

We address some of these scenarios by executing SPARQL queries on our dataset later in the section (cf. Section 5.2.2) but before that we describe our methodology in the following subsection.
5.2.1 Methodology

In this section, we describe our methodology to extract information from various data sources and integrate them. The concept of our approach is also depicted in Figure 5.2, which basically covers the layers as described in the following:

1. The software project and software developer’s information from different open source repositories are extracted and transformed into RDF, yielding RDF datasets.

2. Interlink the RDF datasets with each other and across different data sources, where necessary.

3. Load the interlinked RDF datasets into a SPARQL endpoint, which enables software stakeholders to query the interlinked data sources in order to address different use case scenarios.

Figure 5.2: A conceptual diagram showing the transformation and integration of software developers and software projects related information from different open source repositories.

5.2.1.1 Transforming Data Sources into RDF

In order to transform the information contained inside different data sources into RDF datasets, we consider Apache Ant software project repositories, GitHub and Ohloh as our example data sources. Although, our approach can be applied to other data sources as well. We generate RDF datasets for mailing list archives, bug repository, source control repository and source code of Apache Ant software project. In order to generate and interlink information contained inside Apache Ant software project repositories, we follow a Linked Data driven approach outlined in Section 3.2.3 and Section 3.2.4. Moreover, we extract software project and software developer
related information contained inside GrtHub and OHloh and convert them to RDF datasets based on the same principles outlined in Section 3.2.3.

**PUBLISHING GITHUB AS RDF DATASET**

As mentioned in the previous section, GrtHub dataset collected by FLOSSmole community provides only meta information about software projects. However, we are interested in detailed information about software projects and software developers. Therefore, we developed our own scripts to crawl software projects and software developers related information from GrtHub. GrtHub provides an API access over HTTPS and allows to send and receive data as JSON. We used the API to retrieve software projects and software developers information in JSON and converted it into RDF datasets. We do not go into details of the metadata provided by API but recommend interested readers to have a look at their API tutorial\(^{26}\). In the following, we explain how to publish information about a software project and software developer that is contained in GrtHub as an RDF dataset based on the principles that are outlined in Section 3.2.3. The same approach can be applied to the information contained inside other code forges.

**P1** Identify entities that exists on GrtHub

In order to represent the information contained inside a code forge (e.g., GrtHub) as RDF, we are required to identify different entities that it contains. As a code forge hosts software projects that are developed by software developers, therefore we consider software project and software developer as entities for GrtHub. A software project also contains various software artifacts (e.g., source code, source control commits, bug reports etc.), which are also considered as entities but we do not consider them here as we have already explained how to represent those software artifacts as RDF datasets in Section 3.2.3.

**P2** Assign globally unique identifiers to these entities

Once entities are identified in GrtHub, we assign globally unique identifiers (i.e., HTTP URIs) to these entities so that useful information about each entity can be provided upon request. As an example, we present most commonly found entities (in GrtHub) and their exemplary HTTP URI patterns in Table 5.10. Most code forges do possess similar kind of information, therefore, similar kind of entities can be easily identified in other code forges.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
<th>Example URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>Apache Ant</td>
<td><a href="http://example.org/github/project/ant">http://example.org/github/project/ant</a></td>
</tr>
<tr>
<td>developer</td>
<td>Stefan Bodewig</td>
<td><a href="http://example.org/github/author/bodewig">http://example.org/github/author/bodewig</a></td>
</tr>
</tbody>
</table>

Table 5.10: URI patterns for different types of entities found in GrtHub.

In order to assign globally unique identifiers to a software project, we use `code forge name` in the URI (e.g., we have used “github” in the URIs in Table 5.10) in order to avoid conflicting URIs if different code forges host software projects that have similar names. As a software developer often work on multiple software projects, therefore, we also use `code forge name` in the URI of a software developer in order to keep track of the development activities of a software developer within a particular code forge. Note that the domain name (i.e., `example.org`) used in the URI patterns listed

---

in Table 5.10 is for illustration purposes only and must be replaced by the domain name of a data publisher.

**p3 Describe entities and relationships between them using RDF vocabularies**

Once the HTTP URIs are assigned to the entities in GitHub, the next step is to define the relationships between them. A software project at GitHub usually contains basic information about itself, core software developers of a software project, software developers who have forked the project (i.e., contributors of that software project) and source control commit logs relevant to that software project. Therefore, we can define the relationships between a software project with its core software developers, contributors and source control commit logs. RDF enables us to define these relationships explicitly and publishes this information in a form so that other entities can discover and link to it for further use. In order to show how RDF transformation of a GitHub software project may look like, we provide an example in Listing 5.10.

```
@prefix lf: <http://vocab.deri.ie/linkedfloss#> .
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix baetle: <http://baetle.googlecode.com/svn/ns/> .

<http://example.org/github/project/ant> a doap:Project;
lf:forkedby <http://example.org/github/author/terabyte>;
lf:watchers <http://example.org/github/author/bodewig>;
doap:description "Apache Ant";
doap:name "ant";
doap:programming-language "Java";
doap:developer <http://example.org/github/author/bodewig>;
doap:repository "git://github.com/apache/ant.git";
lf:commits <http://example.org/github/commits/0035b01af>,
<http://example.org/github/commits/3bf8ef5f4>.

<http://example.org/github/commits/0035b01af> a baetle:Committing;
baetle:author "stefan.bodewig";
baetle:summary "simple build file to create ..."
```

Listing 5.10: RDF representation of a software project information extracted from GitHub.

In Listing 5.10, the relationship between a software project and its core software developers is represented as RDF statement using doap:developer property, which tells that a user with ID "bodewig" (represented in the form of URI as: `http://example.org/github/author/bodewig`) is a developer of Apache Ant (represented in the form of URI as: `http://example.org/github/project/ant`). Similarly, contributors who have forked Apache Ant project is represented as RDF statement using lf:forkedby property. Moreover, we have used several other properties from different RDF vocabularies in order to describe meta information (name, programming language, repository URL, description etc.) of a GitHub software project, as shown in Listing 5.10.

A software developer profile at GitHub usually contains basic information about himself/herself, software developers he/she is following or being followed by others and the software projects he/she is developing. We define the relationship of a software developer with other software developers and software projects through RDF statements. To show how RDF transformation of a GitHub software developer profile may look like, we provide an example in Listing 5.11.

In Listing 5.11, the relationship between a software developer and software projects he/she is developing is represented using lf:repo property, which tells that a user with ID "bodewig" (represented in the form of URI as: `http://example.org/github/author/bodewig`) is a developer
of Apache Ant and ANT4NANTANDMSBUILD software projects. Similarly to Listing 5.10, we have used several properties from other RDF vocabularies to describe meta information (name, account-name, homepage, blog, company etc.) of a GitHub software developer, as shown in Listing 5.11.

**PUBLISHING OHLOH AS RDF DATASET**

In order to use the RDF description of software projects and software developers published by RDFOHLOH, we looked into the RDFOHLOH dataset and found that it provides information less than that is available in OHLOH. Hence, we developed our own script to crawl information about software projects and software developers. OHLOH provides a RESTful API to the OHLOH open source directory and returns XML-formatted data in response to HTTP GET requests. We used the OHLOH API to get statistical information about software projects and software developers in XML format and later converted it into RDF datasets. For details on the information provided by OHLOH API, we recommend interested readers to have a look at their API tutorial. In the following, we explain how to publish information about a software project and software developer that is contained inside OHLOH as an RDF dataset based on the principles that are outlined in Section 3.2.3.

**P1** Identify entities that exists on OHLOH

As OHLOH contains statistical information about software projects and software developers, therefore we consider software project and software developer as entities.

**P2** Assign globally unique identifiers to these entities

We assign globally unique identifiers (i.e., HTTP URIs) to the entities that are identified in OHLOH so that useful information about each entity can be provided upon request. As an example, we present most commonly found entities in OHLOH and their exemplary HTTP URI patterns in Table 5.11.

Note that the domain name (i.e., example.org) used in the URI patterns listed in Table 5.11 is for illustration purposes only and must be replaced by the domain name of a data publisher.

**P3** Describe entities and relationships between them using RDF vocabularies
Table 5.11: URI patterns for different types of entities found in OHLOH.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
<th>Example URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>Apache Ant</td>
<td><a href="http://example.org/ohloh/project/ant">http://example.org/ohloh/project/ant</a></td>
</tr>
<tr>
<td>developer</td>
<td>Stefan Bodewig</td>
<td><a href="http://example.org/ohloh/author/bodewig">http://example.org/ohloh/author/bodewig</a></td>
</tr>
</tbody>
</table>

Once the HTTP URIs are assigned to the entities in OHLOH, the next step is to define the relationships between them. A software project at OHLOH usually contains statistical information about the number of users who uses a software project, total number of lines of code, software developers contributed to a software project as well as their contributions to that software project based on different programming or scripting languages. Therefore, we can define these relationships explicitly through RDF statements. In order to show how RDF transformation of a software project at OHLOH may look like, we provide an example in Listing 5.12.

```
@prefix : <http://example.org/ohloh/project/ant#> .
@prefix lf: <http://vocab.deri.ie/linkedfloss#> .
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:15491947040395 a doap:Project; lf:users "1174";
  lf:loc "218797";
  doap:developer :15491947040395;
  doap:name "Apache Ant"
.
:15491947040395 a foaf:Person;
  lf:commits "3994";
  lf:language_fact :15491947040395_5 , :15491947040395_3;
  foaf:accountName "bodewig";
  foaf:name "Stefan Bodewig"
.
:15491947040395_5 lf:commits "2501";
  lf:language-id "5";
  doap:programming-language "Java"
.
:15491947040395_3 lf:commits "1311";
  lf:language-id "3";
  doap:programming-language "XML"
.
```

Listing 5.12: RDF representation of a software project information extracted from OHLOH.

In Listing 5.12, the relationship between a software project and its core software developers is represented as RDF statement using doap:developer property, which tells that a user with ID "15491947040395" (represented in the form of URI as: http://example.org/ohloh/project/ant#15491947040395) is a developer of Apache Ant (represented in the form of URI as: http://example.org/ohloh/project/ant). Moreover, the contributions of a software developer based on different programming or scripting languages are represented as RDF statements using lf:language_fact property. Each contribution is further described using different properties. For example, lf:commits property is used to describe total commits made by a software developer using a specific programming language.
A software developer profile at OHLOH usually contains basic information about himself/herself and the software projects he/she is developing. To show how RDF transformation of a software developer information available at OHLOH may look like, we provide an example of RDF representation in Listing 5.13.

```xml
@prefix lf: <http://vocab.deri.ie/linkedfloss#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .

http://example.org/ohloh/author/bodewig a foaf:Person;
lf:kudo-rank "9";
lf:location "Monchengladbach, Germany";
foaf:based_near [geo:lat "51.191261";
geo:long "6.442066"];
foaf:name "Stefan Bodewig";
lf:repo <http://example.org/ohloh/project/ant>,
<http://example.org/ohloh/project/Ant4NantAndMSBuild>,
<http://example.org/ohloh/project/xmlunit> .
```

Listing 5.13: RDF representation of a software developer profile extracted from OHLOH.

In Listing 5.13, the relationship between a software developer and software projects he/she is developing is represented using lf:repo property, which tells that a user with ID “bodewig” (represented in the form of URI as: http://example.org/ohloh/author/bodewig) is a developer of APACHE ANT, ANT4NANTANDMSBUILD and XMLEUNIT software projects.

### 5.2.1.2 Interlinking RDF Data Sources

From the listings (cf. Listing 5.10, Listing 5.11, Listing 5.12, Listing 5.13), we are able to conclude that the RDF fragments are talking about two different entities, i.e., a software project named “APACHE ANT” and a software developer named “Stefan Bodewig”. While manually inspecting the RDF datasets of GOOGLECODE and SOURCEFORGE code forges, we found that “Stefan Bodewig” is also involved in some software projects on these code forges. Thus, we interlink these RDF fragments using an owl:sameAs property indicating that these URIs actually refers to the same entity (see Listing 5.14). We do not provide any algorithm or heuristic to automatically interlink software developer IDs in this section but rather focus on showing the advantages of interlinking software developer information across different open source software repositories on the Web using Linked Data principles. Through interlinking APACHE ANT software repositories, GitHub, GOOGLECODE, SOURCEFORGE and OHLOH RDF data sources based on the IDs of “Stefan Bodewig”, we are able to query the software projects, which he is developing at different code forges, his development activity (e.g., last month commits, bug fixes, social interactions etc.) on these software projects as well as statistical information about his development based on different programming languages (available via OHLOH).

Establishing owl:sameAs links between the software developers URIs at OHLOH with APACHE or GitHub software developers URIs is pretty straight forward. OHLOH computes statistics of a particular software project by analyzing source control repositories, which means that the software developer IDs associated to a software project at OHLOH will be same as the software developer IDs in the source control repository of that particular software project. Therefore, in order to generate a set of owl:sameAs links between APACHE ANT and its relevant OHLOH software project, we extract a list of software developer IDs who have committed on the source control repository
of Apache ANT software project by querying the source control RDF datasets. Further, we query the Ohloh RDF dataset to retrieve a list of software developer IDs for Apache ANT software project. Finally, we compare both lists of software developer IDs using string similarity algorithm in order to generate owl:sameAs links between them. There are 46 software developers who have committed source code to the source control repository of Apache ANT software project from the beginning of the software project to date. Among them, we found 45 software developer IDs in the Ohloh dataset. Therefore, we establish 45 links between software developer URIs that exists on a source control RDF dataset of Apache ANT and the software developer URIs of Ohloh software project RDF dataset that exists on Ohloh. By doing so, we are able to query the activities of a particular software developer in Apache ANT software project repositories as well as query Ohloh dataset to retrieve the development ratio of that particular software developer based on different programming languages that are used in a software project.

5.2.2 Preliminary Findings

In order to show the benefits of integrating software developer related information from different data sources, we loaded RDF datasets of GitHub, Ohloh and Apache software projects into a SPARQL endpoint. We use the SPARQL endpoint to execute SPARQL queries that are presented in this section. We start with a simple query to list all software projects on which a software developer is working on or has worked in the past (cf. Listing 5.15).

```
PREFIX doap: <http://usefulinc.com/ns/doap#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

select distinct ?projects
WHERE {
  {?dev owl:sameAs <http://example.org/ant/author/bodewig> .}
  UNION
  {<http://example.org/ant/author/bodewig> owl:sameAs ?dev .}
}
```

Listing 5.15: A query that lists all software projects on which a software developer of Apache ANT is working in different code forges.

Considering that we have owl:sameAs links between similar software developer IDs (cf. Listing 5.14) in the SPARQL endpoint, the query returns all software projects on which “Stefan Bodewig” is working across different code forges. The results of the above query is shown in Table 5.12.
As shown in Table 5.12, we have a list of software projects on which a software developer is working across different code forges. Now, we are able to query the actual contributions that have been made on these software projects by that particular software developer. In our current set up, the FLOSSMole dataset only contains meta information about software projects, hence we are not able to retrieve contributions made to the software projects hosted on SourceForge and GoogleCode. Given that the Ohloh dataset contains development statistics of a software developer for different programming languages, therefore we query the number of commits he had made to a particular software project using different programming languages as shown in Listing 5.16.

```
PREFIX lf: <http://vocab.deri.ie/linkedfloss#>
PREFIX doap: <http://usefulinc.com/ns/doap#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

select ?commits ?language
WHERE {
  ?id foaf:accountName "Stefan Bodewig" .
  ?repo doap:name "Apache Ant" .
}
```

Listing 5.16: A SPARQL query that retrieves number of commits made to a particular software project by a software developer using different programming languages.

The results of the SPARQL query (cf. Listing 5.16) is shown in Table 5.13, which makes it easier to understand the contributions made by a software developer using different programming languages, based on the number of commits he had made to a software project. For example, result shows that the software developer has made most contributions in “Java” in comparison to “MetaFont” programming language based on his commits to the source control repository.

It is likely that a software developer has contributed to several open source software projects that are indexed by Ohloh. Therefore, we are also able to query the development statistics for
all programming languages, which a software developer has used to contribute to different open source software projects. The query in Listing 5.17 returns an average commit ratio of a software developer for different programming languages based on all software projects to which he has contributed. The results returned by the query (cf. Listing 5.17) gives an idea about the expertise level of a software developer in different programming languages, which is shown in Table 5.14.

### Table 5.13: List of source control commits made to a particular software project by a software developer using different programming languages.

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>2501</td>
</tr>
<tr>
<td>HTML</td>
<td>1404</td>
</tr>
<tr>
<td>XML</td>
<td>1311</td>
</tr>
<tr>
<td>JavaScript</td>
<td>257</td>
</tr>
<tr>
<td>Shell Script</td>
<td>74</td>
</tr>
<tr>
<td>XSL Transformation</td>
<td>30</td>
</tr>
<tr>
<td>DOS Batch Script</td>
<td>28</td>
</tr>
<tr>
<td>CSS</td>
<td>23</td>
</tr>
<tr>
<td>Perl</td>
<td>8</td>
</tr>
<tr>
<td>Python</td>
<td>7</td>
</tr>
<tr>
<td>C#</td>
<td>6</td>
</tr>
<tr>
<td>XML Schema</td>
<td>1</td>
</tr>
<tr>
<td>MetaFont</td>
<td>1</td>
</tr>
</tbody>
</table>

Listing 5.17: A SPARQL query that retrieves average commit ratio of a software developer for different programming languages based on his development activity in different software projects.

Apart from that, we are also able to query the total number of commits made by all software developers to a software project using different programming languages. This sort of information helps newcomers (i.e., volunteers) to get useful insights into different programming or scripting languages, which they can potentially use to start contributing to an open source software project.

In this section, we have argued with examples on the benefit of integrating software developer related information across different data sources in order to serve different use cases often found in the software development domain. In fact, most of the questions pointed out by Conklin (cf. Section 5.1) can now easily answered by simple queries given that the data sources are interlinked.
Table 5.14: Average commit ratio of a software developer on different programming languages.

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Commits Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>327.00</td>
</tr>
<tr>
<td>JavaScript</td>
<td>257.00</td>
</tr>
<tr>
<td>HTML</td>
<td>171.88</td>
</tr>
<tr>
<td>XML</td>
<td>129.57</td>
</tr>
<tr>
<td>Shell Script</td>
<td>37.50</td>
</tr>
<tr>
<td>DOS Batch Script</td>
<td>28.00</td>
</tr>
<tr>
<td>C#</td>
<td>27.00</td>
</tr>
<tr>
<td>CSS</td>
<td>11.33</td>
</tr>
<tr>
<td>XSL Transformation</td>
<td>8.00</td>
</tr>
<tr>
<td>Perl</td>
<td>8.00</td>
</tr>
<tr>
<td>Python</td>
<td>7.00</td>
</tr>
<tr>
<td>XML Schema</td>
<td>1.50</td>
</tr>
<tr>
<td>MetaFont</td>
<td>1.00</td>
</tr>
<tr>
<td>Ruby</td>
<td>1.00</td>
</tr>
</tbody>
</table>

5.3 Conclusion

In this chapter, we have motivated and proposed integration of code forges based on their metadata, similar software projects and software developers. We argue that a common model definition is required to enable integration across different code forges. Therefore, we proposed and supported with examples that Semantic Web technologies (Linked Data in particular) allow integrating knowledge not only across different code forges but also to other relevant data sources available on the Web. We considered only SOURCEFORGE, GOOGLECODE and GITHUB for modeling, however there are several other code forges available on the Web. Therefore, one potential future work direction is to extend the common model definition by taking into account other code forges.

We have made some initial steps towards realizing this whole integration vision, although more work needs to be done in this direction. We have demonstrated how to achieve interlinking across different code forges based on similar software projects and software developers using Linked Data approach but not much information was available to query due to the limited data available for each code forge, which is provided by the FLOSSMOLE community. FLOSSMOLE only provides metadata information about software projects, therefore we were not able to query the software artifacts underlying different software repositories in order to discover valuable information. However, if in the future FLOSSMOLE publish software artifact level information of every software project for different code forges then it can be easily converted to RDF datasets (as discussed in detail in Section 3.2.3) and further integration (as discussed in Section 3.2.4) can be achieved. In order to demonstrate the real benefits of integrating code forges to each other, we are required to extract detailed information relevant to a software project, which are hidden deep inside software repositories of code forges. This would allow interested users to find answers to various questions:

- how much percentage of development is carried out on a software project across different code forges?
- what is the ratio of contributions of a software developer on different software projects across code forges?
• how big the community is around a particular software project across different code forges?

Apart from code forges, we also considered statistical service that provides statistical information about software projects. We motivated and further supported with examples that integrating software project and software developer related information across different open source software repositories enable software stakeholders to not only query statistical information about a software project or software developer but also allow them to trace software developer’s development activities across different software projects as well as information that is hidden deep inside software repositories hosting a particular software project.

In this chapter, we considered a small number of software projects and software developers in order to show the benefits of integration through exemplary SPARQL queries and scenarios. Moreover, we did not provide a real time environment where software project and software developer related information from code forges and other relevant data sources (e.g., statistical services) are continuously extracted, published and integrated. However, we believe that this is particularly important and will open new research directions to synchronise and integrate such large data sources.
n the previous chapter, we motivated and proposed integration of code forges based on metadata, similar software projects and software developers. Further, we presented integration of software developer related information across different open source software repositories and demonstrated its benefit through some SPARQL queries. Referring to Figure 1.1, we have covered Linked Data based integration of software repositories relevant to a software project, code forges and statistical services. Further, we demonstrated the benefits of integration through some use case scenarios. In this chapter, we focus on social media channels and provide evidence that software project and software developer related information also exists on social media channels. In particular, we highlight the needs of integrating social media channels and software repositories relevant to a software project.

The growing interest in the usage of online social media channels such as Facebook, Twitter, MySpace, LinkedIn etc., have attracted millions of users. Users join social media channels to share information about their activities and opinions on various topics. Topics usually range from daily routine activities to current events, media news etc. Much research has been carried out on analyzing these enormous social media channels with different perspectives in mind – such as, privacy issues [Fang and LeFevre, 2010; Rosenblum, 2007], influential users within a community [Agarwal et al., 2008; Trusov et al., 2010; Weng et al., 2010], churns in online social communities [Karnstedt et al., 2010] etc. With the popularity of social media channels, we have witnessed extensive usage of these platforms by the open source community and software developers in order to disseminate software project related information to a wider audience. Software developers often discuss or share their thoughts and opinions about a particular release or issues in software projects on these platforms. Despite the emerging growth in the usage of social media channels by open source community, its usage hasn’t been studied in depth [Yang et al., 2013].

In the last few years, research has started to emerge on analyzing the activities of software developers with respect to software projects on social media channels. [Black et al., 2010] carried out a survey to explore whether social media channels are being used by software developers. In their survey, respondents revealed that they use social media to communicate with their colleagues, ranking Twitter and Instant Messaging to be the most popular media for their communication. In another work, [Black and Jacobs, 2010] investigated the use of social media by software developers while developing software projects and its impact on software quality. [Wang et al., 2013] investigated how open source community uses social media channels through studying the usage of Twitter by software developers of Drupal open source content management system. In another study, [Bougie et al., 2011] found that software developers extensively leverages Twitter’s capabilities for conversation and information sharing. [Tian et al., 2012] performed a preliminary study on what software developers microblogging about by analyzing the content of microblogs for Twitter and further categorize them. Similarly, [Prasetyo et al., 2012] automatically classified tweets based on their relevance to software engineering. Last but not least, [Singer et al., 2013]
discovered in their qualitative study that software developers use Twitter to keep themselves up-to-date with the fast changing development landscape, for learning and building relationships.

Given the interests in the usage of online social media channels by software developers, we study how software developers use social media channels (Twitter in particular) by harvesting their software project related activities. Based on our findings, we highlight the needs of integrating software repositories and social media channels (i.e., interlinking software project related tweets/posts/hashtags, software developer IDs, software project IDs etc.) in order to get an integrated view on the software project from the development and social aspects.

The remainder of this chapter is organized as follows:

- In Section 6.1, we motivate integration of social media channels and software repositories relevant to a software project. We consider Twitter as an example of social media channel in this chapter and provide some examples of converting Twitter dataset to RDF. Further, we study the usage of Twitter by software developers through harvesting their software project related activities in our crawled Twitter dataset. In particular, we present the most frequently used hashtags by software developers and further investigate if software project related hashtags are the most frequent and commonly used hashtags by software developers.

- In Section 6.2, we study the social interaction behavior of software developers with each other across different communication channels. In particular, we study the social interactions happened among software developers on Twitter and compare it with their interactions happened on the mailing list of a software project. Based on this study, we investigate if there is a correlation between software developers interaction with each other across different communication channels.

- Section 6.3 concludes with lessons learnt from this research.

6.1 INTEGRATING SOCIAL MEDIA CHANNELS AND SOFTWARE PROJECT REPOSITORIES

The growing interest in the usage of online social media channels (e.g., Facebook, Twitter, LinkedIn etc.) have attracted the open source software community. Open source software projects are often found to adopt an identity on these social media channels (e.g., Apache Solr/Lucene\(^1\) on Twitter, MySQL\(^2\) on Facebook) in order to disseminate software project related information (release announcements, major bug fixes etc.) or gather feedback/questions posted by the users. Software developers contributing to open source software projects also exists on social media channels. Quite often, they discuss, debate or share experiences with others relevant to a software project using hashtags (e.g., #apache, #maven, #hadoop etc.). For example, software developers often communicate technical details relevant to a software project’s architecture, source code or bugs as well as any announcements (see Figure 6.1) or experiences\(^3\) relevant to a software project. Hence, the discussions covering open source software projects are not limited to dedicated forums or mailing lists, there also exists huge amount of information on the social media channels.

Given the information relevant to a software project that are shared by software developers on social media channels, we cannot ignore the importance of consuming this information. Moreover,

---

1 https://twitter.com/SolrLucene
2 https://www.facebook.com/mysql
3 https://twitter.com/olamy/status/231031288734285824
it is worth mentioning that the information related to open source software projects and software developers are distributed on the Web in heterogeneous data islands i.e., social media channels and software repositories of a software project as shown in Figure 6.2.

Hence, there is a need to bridge the connection between software repositories and social media channels (i.e., interlinking software project related tweets/posts/hashtags, software developer IDs, software project IDs etc.), as it has been previously proposed by [Guzzi et al., 2010; Reinhardt, 2009; Treude et al., 2011] to integrate microblogging with software development. By enabling this connection, we have an integrated view on the software project that can be exploited to support certain use case scenarios:

- End-users response on a particular release of a software project.
- Popularity of a software project by applying sentiment analysis [Hu and Liu, 2004] on social messages (i.e., tweets, posts etc.).
- Tracing software developer’s social activity related to a software project.
- Analyzing the social interactions of software developers with each other across different communication channels (i.e., social media channels and software repositories).

We consider the social interactions of software developers with each other across different communication channels as a case study in the next section. In this section, we investigate how software developers use Twitter by conducting an exploratory study that analyses their Twitter accounts. This study will help us to understand if software developers promote or share news relevant to software projects on social media channels (Twitter in particular).

6.1.1 Transforming Twitter Data into RDF

In order to carry out our study, we are required to crawl data from Twitter. Moreover, we are required to publish Twitter in a format so that we can query the dataset easily. Therefore, in this section we describe our methodology to extract software developer’s profile and his/her tweets from Twitter and transform them into RDF datasets. Twitter offers an Application Programming Interface (API)\(^4\), which makes it easy to crawl and collect data from Twitter. We crawl

\(^4\) https://dev.twitter.com/docs
software developers public profiles and their tweets using Twitter API and later transform it to RDF using our custom written scripts.

6.1.1.1 Publishing User Profile as RDF Dataset

A public user profile in Twitter includes the full name, a short biography, the location, web page and the number of tweets posted by a user. The people who are followed by a user and the people who follow a user is also listed. In the following, we explain how to publish a Twitter user profile as an RDF dataset based on the principles that are outlined in Section 3.2.3.

p1 Identify entities that exists in a Twitter profile

As a Twitter profile is focused on a single user, therefore there is only one entity (software developer) we identify in this case.

p2 Assign globally unique identifiers to these entities

Once entities are identified in a Twitter user profile, we assign globally unique identifiers to these entities. In particular, we assign HTTP URIs to these entities so that useful information about each entity can be provided upon request. As an example, we present an entity and its exemplary HTTP URI pattern in Table 6.1.

Note that the domain name (i.e., example.org) used in the URI pattern listed in Table 6.1 is for illustration purposes only and must be replaced by the domain name of a data publisher.
Table 6.1: URI pattern for an entity found in a Twitter user profile.

### p3 Describe entities and relationships between them using RDF vocabularies

Once the HTTP URIs are assigned to the entities in a Twitter user profile, the next step is to define the relationships between them. A user profile contains information about other users who are following that particular user and also information about users who are being followed by that particular user. Therefore, we can define the relationships between a user and his/her followers and followings. RDF enables us to define these relationships explicitly and publishes this information in a form so that other entities can discover and link to it for further use. In order to show how RDF transformation of a Twitter user profile may look like, we provide an example of a Twitter user profile in Listing 6.1.

```plaintext
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix lf: <http://vocab.deri.ie/linkedfloss#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix doap: <http://usefulinc.com/ns/doap#> .
<http://example.org/twitter/brettporter> a foaf:Person ;
foaf:accountName "brettporter" ;
foaf:name "Brett Porter" ;
foaf:homepage <http://twitter.com/brettporter> ;
lf:followers "1015" ;
lf:following "648" ;
lf:status_count "6110" ;
dct:created "2007-03-26T00:05:50" ;
lf:location "Sydney, Australia" ;
doap:blog "http://brettporter.wordpress.com" ;
dct:language "en" ;
foaf:knows <http://example.org/twitter/aheritier> .
```

Listing 6.1: RDF representation of a software developer’s Twitter profile.

In Listing 6.1, the relationship between a user and his follower is represented as RDF statement using foaf:knows property, which tells that a user with ID “brettporter” (represented in the form of URI as: http://example.org/twitter/brettporter) knows another user with ID “aheritier” (represented in the form of URI as: http://example.org/twitter/aheritier). Moreover, we have used several other properties from different RDF vocabularies to describe profile information (name, homepage, blog, description, location etc.) of a Twitter user.

### 6.1.1.2 Publishing Tweet as RDF Dataset

A tweet is a message with a limit of 140 characters, which is shared on Twitter by a user and can be seen by anyone unless the profile of a user is kept private. A tweet sent by a user could be on any topic ranging from daily routine activity to current events, news stories etc. Directed conversations among users on Twitter typically involves usage of an “@” symbol followed by the username to address messages to them.
Identify entities that exists in a tweet

In order to represent tweet in RDF, we identify different entities that exists in a tweet. As a user often mention other users in a tweet using "@" symbol, therefore we consider tweet ID and user as entities.

Assign globally unique identifiers to these entities

After identifying entities in a tweet, we assign globally unique identifiers to these entities. In particular, we assign HTTP URIs to these entities so that useful information about each entity can be provided upon request. As an example, we present entities and their exemplary HTTP URI patterns in Table 6.2.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
<th>Example URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>327581901932085249</td>
<td><a href="http://example.org/twitter/status/327581901932085249">http://example.org/twitter/status/327581901932085249</a></td>
</tr>
<tr>
<td>developer</td>
<td>Brett Porter</td>
<td><a href="http://example.org/twitter/brettporter">http://example.org/twitter/brettporter</a></td>
</tr>
</tbody>
</table>

Table 6.2: URI patterns for different types of entities found in a tweet.

Note that the domain name (i.e., example.org) used in the URI patterns listed in Table 6.2 is for illustration purposes only and must be replaced by the domain name of a data publisher.

Describe entities and relationships between them using RDF vocabularies

Once the HTTP URIs are assigned to different entities in a tweet, the next step is to define the relationships between them. A tweet contains information about its creator, therefore, we can define the relationship between a tweet and his/her creator. Moreover, a tweet usually contains mention of other users, therefore we can also explicitly specify this relationship. These statements of relationships are, in essence, links between entities that can be easily expressed explicitly using RDF. In order to show how RDF transformation of a tweet may look like, we provide an example in Listing 6.2.

```
@prefix siocct: <http://rdfs.org/sioc/types#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .
@prefix moat: <http://moat-project.org/ns#> .

https://example.org/twitter/status/327581901932085249 a siocct:MicroblogPost;
  dct:created "2013-04-26T01:36:22";
  dct:creator <http://example.org/twitter/brettporter>;
  sioc:content "@ulander if you’re at #RightScaleCompute, make sure to catch up
  with the other guys from @MaestroDev!";
  sioc:id "327581901932085249";
  sioc:mentions <http://example.org/twitter/ulander>,
              <http://example.org/twitter/MaestroDev>;
  moat:taggedWith "RightScaleCompute"
.
```

Listing 6.2: RDF representation of a tweet.

After transforming the data sources to RDF, we loaded the RDF datasets into a SPARQL end-point. In this chapter, we do not focus on interlinking software developer and software project
related tweets from Twitter to the various software artifacts (i.e., bug, email, commit ID, source code, software developer ID etc.) contained inside different software repositories. Hence, we do not present any approach on creating owl:sameAs links between relevant entities across different data sources. Different approaches [Conklin, 2007; Iqbal and Hausenblas, 2012, 2013; Robles and Gonzalez-Barahona, 2005] can potentially be utilized in order to achieve the interlinking between Twitter datasets and various software artifacts but it is not in the current scope of this work.

6.1.2 Preliminary Study

[Singer et al., 2013] find out in their survey that the participants (who took part in the survey) follow specific software developers or software projects that are of their interest or relevant to their work. They find it extremely important as it enables a one to one connection with software developers that can provide latest information about software projects, which would have been difficult to obtain otherwise. Therefore, in this section we investigate if software developers are using Twitter as a platform to promote or share news relevant to the software projects that they are developing. We investigate the usage of Twitter through analysing quantitative information on some selected software developers Twitter accounts. We devise 3 questions that revolves around the usage of software project related hashtags by software developers on Twitter. In particular, we study the usage of hashtags by software developers and investigate if software project related hashtags are most commonly and frequently used by software developers. We believe that disseminating software project related information on social media channels like Twitter helps them to collect quick response from a large community and also attract contributors for the software project. In our study, we do not analyze the content of tweets but perform a quantitative analysis based on the usage of software project related hashtags by software developers of some selected Apache projects.

Our crawled Twitter dataset comprises of tweets from software developers of Apache projects. The randomly selected Apache software projects and its associated software developers used in this study are shown in Table 6.3. For each Apache software project under consideration, we considered only those software developers who have commit rights on the source control repository. For these software developers, we manually checked if they also exist on Twitter and using the Twitter account frequently. We found some software developers who do exist on the Twitter platform but tweeted very little (≈10-15 tweets only) and some software developers have set their profiles as private so that only authorized followers can view their tweets. We ignored such software developers in the Twitter data crawling, RDF transformation and analysis phase. Table 6.3 shows for each Apache software project, the number of software developers who have made commits to the source control repository and the software developers actually found on Twitter. Although, not all software developers developing the Apache software projects (under consideration for this study) are found on Twitter but the results in Table 6.3 shows good evidence of the existence of software developers on Twitter, which will be sufficient to address some questions in this section.

For each software developer, we obtained their tweets and profile information using Twitter API. Later, we transformed the crawled data into RDF datasets (cf. Listing 6.1 and Listing 6.2) and loaded it into SPARQL endpoint. Table 6.4 presents the total number of software developers, total number of tweets and distinct hashtags used in the tweets.

We crawled data from January 2009 till January 2013. The distribution of tweets per software developer is shown in Figure 6.3. The average number of tweets each software developer posted is
Table 6.3: Software developers contributing to Apache software projects and also using Twitter.

<table>
<thead>
<tr>
<th>Apache Projects</th>
<th>Software Developers (SVN)</th>
<th>Software Developers (Twitter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Camel(^a)</td>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>Apache Directory(^b)</td>
<td>51</td>
<td>11</td>
</tr>
<tr>
<td>Apache Felix(^c)</td>
<td>47</td>
<td>17</td>
</tr>
<tr>
<td>Apache Hadoop(^d)</td>
<td>97</td>
<td>35</td>
</tr>
<tr>
<td>Apache Logging(^e)</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>Apache Lucene(^f)</td>
<td>51</td>
<td>18</td>
</tr>
<tr>
<td>Apache Maven(^g)</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>Apache Mina(^h)</td>
<td>28</td>
<td>9</td>
</tr>
<tr>
<td>Apache MyFaces(^i)</td>
<td>82</td>
<td>16</td>
</tr>
<tr>
<td>Apache OfBiz(^j)</td>
<td>25</td>
<td>11</td>
</tr>
</tbody>
</table>

\(^a\) http://camel.apache.org/
\(^b\) http://directory.apache.org/
\(^c\) http://felix.apache.org/
\(^d\) http://hadoop.apache.org/
\(^e\) http://logging.apache.org/
\(^f\) http://lucene.apache.org/
\(^g\) http://maven.apache.org/
\(^h\) http://mina.apache.org/
\(^i\) http://myfaces.apache.org/
\(^j\) http://ofbiz.apache.org/

Table 6.4: Twitter dataset summary.

<table>
<thead>
<tr>
<th>Software Developers</th>
<th>Tweets</th>
<th>Hashtags</th>
</tr>
</thead>
<tbody>
<tr>
<td>158</td>
<td>186,709</td>
<td>20,395</td>
</tr>
</tbody>
</table>

1181.70. The decline in the curve shows that certain software developers tweeted less on Twitter. One reason could be that certain software developers joined late on Twitter than other software developers. We queried our Twitter dataset in order to find out the oldest and recent joining date of software developers on Twitter. We found that the oldest joining date for a software developer was 2007-03-07 and the recent joining date for a software developer was 2012-04-11. The recent joining date of software developers may be the reason behind less tweets by the software developers on Twitter, however there may be cases where software developers exist on Twitter for a long time but have stopped tweeting shortly after joining Twitter or they may not be the active users of Twitter.

In the following, we now study the usage of Twitter by software developers and investigate some questions based on our crawled Twitter dataset.

Q1: Are technology oriented hashtags most commonly used by software developers?

The usage of hashtags are common among the users of Twitter. A hashtag is a character string preceded by a “#” sign. It is used to categorize the tweets. Hashtags are commonly used by users to specify category, topic or intended audience to a tweet. The usage of hashtags in tweets typically allow others to follow conversations regarding a particular topic. As mentioned
by [Cunha et al., 2011], it is used to not only add context and metadata to tweets but also for promotion and publicity purposes. Software developers also use software project related hashtags to create communities of people (i.e., software developers, contributors or users of the software project) that are interested in a particular software project. For example, “#maven” marks tweet related to the Apache Maven software project. So, any one interested to know what is under discussion on Twitter regarding Apache Maven software project can search for tweets with a “#maven” hashtag. We compute the distribution of hashtags based on the number of times each hashtag appeared in our crawled Twitter dataset and plot the results in Figure 6.4. Moreover, the average number of times a hashtag is used by software developers in their tweets is 2.046.

Based on Table 6.4, we see that software developers have used plenty of hashtags in their tweets. Therefore, we are interested to know most commonly used hashtags by software developers in our crawled Twitter dataset. We retrieve a list of distinct hashtags from the SPARQL endpoint (see query in Listing B.5). Then for each hashtag, we query (see query in Listing B.6) the number of distinct software developers that have used that particular hashtag in his/her tweets. We ignore the frequency of usage of a hashtag in this use case and present only the top 15 most commonly used hashtags in Table 6.5. The ranking of hashtags are done based on the number of software developers that have used a particular hashtag.

The result shows that tweeting about work related stuff is common among software developers. The usage of work related hashtags is quite obvious because we crawled dataset of a specific community of users (i.e., Apache software developers) from Twitter. The top 3 hashtags listed in Table 6.5 is obvious due to the following 3 reasons: (1) the software projects under consideration are Apache software projects; (2) the primary programming language used by the Apache software projects under consideration is Java; and (3) Apache software projects under consideration...
are open source. Most hashtags that are commonly used by software developers are technology-oriented (cf. Table 6.5), which shows that software developers often use Twitter to share work related information with others.

Q2: Do software developers tweet most frequently about software projects?

In the previous use case, we studied the overall distribution and most commonly used hashtags by software developers. In the following, we investigate if software developers are specifically tweeting about software projects they are developing. In other words, we are interested to know if software developers promote or talk about the software projects that they are developing using software project related hashtags on Twitter.

Given that we have a list of hashtags used by software developers, \( \text{Hashtag} = \{h_1, h_2, \ldots, h_n\} \). For each Apache software project under consideration, we first retrieve a list of software developers, \( \text{Dev} = \{d_1, d_2, d_3, \ldots, d_n\} \). Then for each software developer, we extract all occurrences of hashtags in his/her tweets, \( \text{Hashtag}_{d} = \{h_1, h_4, h_1, h_3, h_7, \ldots, h_n\} \), \( \text{Hashtag}_{d} \subseteq \text{Hashtag} \) (see query in Listing B.7). As a software developer may use same hashtag multiple times, therefore we summed all occurrence of similar hashtags, \( \text{Hashtag}_{freq} = \{h_1(2), h_4(1), h_3(1), \ldots, h_n(n)\} \) where \( \text{Hashtag}_{freq} \subseteq \text{Hashtag}_{d} \subseteq \text{Hashtag} \). Later, we summed all occurrences of same hashtags used by all software developers of an Apache software project and rank the results based on most frequently used hashtags.

\[
\sum_{d \in \text{Dev}} \text{Hashtag}_{freq}
\]
### Table 6.5: Most commonly used hashtags by software developers.

<table>
<thead>
<tr>
<th>Hashtags</th>
<th>Software Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache</td>
<td>147</td>
</tr>
<tr>
<td>java</td>
<td>139</td>
</tr>
<tr>
<td>opensource</td>
<td>106</td>
</tr>
<tr>
<td>maven</td>
<td>95</td>
</tr>
<tr>
<td>oracle</td>
<td>86</td>
</tr>
<tr>
<td>hadoop</td>
<td>80</td>
</tr>
<tr>
<td>fb</td>
<td>78</td>
</tr>
<tr>
<td>apachecon</td>
<td>74</td>
</tr>
<tr>
<td>git</td>
<td>73</td>
</tr>
<tr>
<td>android</td>
<td>71</td>
</tr>
<tr>
<td>devoxx</td>
<td>69</td>
</tr>
<tr>
<td>in</td>
<td>68</td>
</tr>
<tr>
<td>cloud</td>
<td>67</td>
</tr>
<tr>
<td>eclipse</td>
<td>62</td>
</tr>
<tr>
<td>scala</td>
<td>61</td>
</tr>
</tbody>
</table>

Although, the hashtags are sorted and ranked based on their usage in tweets by software developers but we select only the top 2 most frequently used hashtags, which we present in Table 6.6.

### Table 6.6: Most commonly used hashtags by software developers. We also show if software developers have ever used software project related hashtags while tweeting.

<table>
<thead>
<tr>
<th>Apache Projects</th>
<th>Software Project Hashtag Used</th>
<th>Most Used</th>
<th>2nd Most Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Camel</td>
<td>✓</td>
<td>#camel</td>
<td>#apache</td>
</tr>
<tr>
<td>Apache Directory</td>
<td>X</td>
<td>#apachecon</td>
<td>#apache</td>
</tr>
<tr>
<td>Apache Felix</td>
<td>✓</td>
<td>#osgi</td>
<td>#camel</td>
</tr>
<tr>
<td>Apache Hadoop</td>
<td>✓</td>
<td>#hadoop</td>
<td>#opendata</td>
</tr>
<tr>
<td>Apache Logging</td>
<td>✓</td>
<td>#rtw2012</td>
<td>#yum</td>
</tr>
<tr>
<td>Apache Lucene</td>
<td>✓</td>
<td>#lucene</td>
<td>#bbuzz</td>
</tr>
<tr>
<td>Apache Maven</td>
<td>✓</td>
<td>#maven</td>
<td>#apache</td>
</tr>
<tr>
<td>Apache Mina</td>
<td>✓</td>
<td>#fb</td>
<td>#netty</td>
</tr>
<tr>
<td>Apache MyFaces</td>
<td>✓</td>
<td>#myfaces</td>
<td>#primefaces</td>
</tr>
<tr>
<td>Apache OfBiz</td>
<td>✓</td>
<td>#rtw2012</td>
<td>#ofbiz</td>
</tr>
</tbody>
</table>

In Table 6.6, we also show if software developers have ever used software project related hashtag on Twitter and found that all Apache software projects that are considered for this study are mentioned by software developers on Twitter using their respective hashtags except Apache Directory. This means that we found no software project related hashtag by the software developers of Apache Directory in our Twitter dataset. We manually checked the usage of Apache Directory hashtag (#ApacheDS) by users on Twitter and found only 6 tweets that contains hashtag for Apache Directory but none of the users mentioning that specific hashtag is a software developer of Apache Directory.
Based on the results in Table 6.6, we see that software project related hashtags are most frequently used by software developers of 5 Apache software projects. We found that hashtag of Apache OfBiz ranked 2nd in the overall usage by software developers of Apache OfBiz but hashtags of Apache Felix, Apache Logging and Apache Mina are ranked quite lower among most frequently used hashtags by software developers of these software projects respectively. We found another interesting thing in the results of Apache Felix where Apache Camel’s hashtag appears to be the 2nd most frequently used hashtag by the software developers of Apache Felix. We investigate it further to identify any possible reason behind this and found that 5 out of the 17 software developers of Apache Felix (cf. Table 6.3) are also software developers of Apache Camel. Therefore, we assume that those 5 software developers tweeted more about Apache Camel than Apache Felix as it is quite obvious from the results.

Q3: How often software developers tweet about software projects?

Previously, we presented the most commonly used hashtags by software developers and further investigated if software project related hashtags are most frequently used by software developers or not. In the results, we found that software project related hashtags are used by software developers of all Apache software projects that are considered except Apache Directory. Based on our findings, we are now interested to know how often software developers tweet about a software project. In particular, we are interested to know if software developers tweet about software projects on daily, weekly or monthly basis.

In order to calculate the average number of days during which a software developer tweets about a software project, we retrieve all tweets mentioning a particular software project using a hashtag, tweets = \{tweet1, tweet2, \ldots, tweet_n\} (see query in Listing B.8 as an example) and extract the timestamp value when it was posted, i.e., date = \{date1, date2, date3, \ldots, date_n\}, where date1 is the timestamp value of tweet1, date2 is the timestamp value of tweet2 and so on. Further, we compute the timestamp difference between two tweets, such that \(\text{day} = \{\text{day}_1, \text{day}_2, \ldots, \text{day}_n\}\), where \(\text{day}_1 = \text{date}_2 - \text{date}_1\), \(\text{day}_2 = \text{date}_3 - \text{date}_2\) and so on. Finally, we compute the average number of days that software developers took in order to tweet about a software project as shown in Table 6.7.

\[
\sum \frac{\text{day}}{|\text{day}|}
\]

In Table 6.7, we present the total tweets in which software project related hashtag is found and the average number of days after which a software project related hashtag is used in a tweet by software developers. The average time of each Apache software project varies due to the variation in the usage of software project related hashtags. We have identified the inverse proportion between hashtags usage and the average time in our study. For example, if hashtags are used often in tweets (i.e., 1,578) then the average time is less (i.e., 0.89) and if hashtags are rarely used in tweets (i.e., 2), then the average time increases (i.e., 50). Moreover, we have ignored keyword-based mentioning of software projects (e.g., mentioning of “hadoop” in the twitter message text) while computing the average time, which may have reduced the average time of hashtags usage in our study. Further, the results we have presented in Table 6.7 is based on a Twitter dataset that contain tweets of only selected software developers of these Apache projects. We haven’t taken into account the usage of software project related hashtag by all users on Twitter for a given time period, which we believe (if computed) will contradict with the average time for each Apache project listed in Table 6.7.
Table 6.7: Average number of days taken by software developers to post a software project related tweet. We also show number of tweets in which software project related hashtag is used by software developers.

<table>
<thead>
<tr>
<th>Apache Projects</th>
<th>Tweets containing Hashtags</th>
<th>Average Time (#days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APACHE CAMEL</td>
<td>475</td>
<td>2.68</td>
</tr>
<tr>
<td>APACHE DIRECTORY</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>APACHE FELIX</td>
<td>22</td>
<td>62.27</td>
</tr>
<tr>
<td>APACHE HADOOP</td>
<td>1,578</td>
<td>0.89</td>
</tr>
<tr>
<td>APACHE LOGGING</td>
<td>20</td>
<td>44.30</td>
</tr>
<tr>
<td>APACHE LUCENE</td>
<td>376</td>
<td>4.38</td>
</tr>
<tr>
<td>APACHE MAVEN</td>
<td>943</td>
<td>1.75</td>
</tr>
<tr>
<td>APACHE MINA</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>APACHE MyFaces</td>
<td>24</td>
<td>31.29</td>
</tr>
<tr>
<td>APACHE OfBiz</td>
<td>62</td>
<td>18.85</td>
</tr>
</tbody>
</table>

In this section, we have investigated the usage of Twitter as a platform by software developers in relation to the software projects they are developing. As our study consists of limited number of software projects, it remains to be seen whether a large number of open source software developers demonstrate similar kind of behavior. Therefore, we can not say that all or majority of software developers frequently tweets about software projects using hashtags. While more research is needed to better understand these and other related questions, our study shows the existence of software project related information on Twitter. Therefore, this information can be integrated with software repositories hosting that particular software project in order to support certain use case scenarios that are mentioned in the beginning of this section.

As we have identified the existence of software developers on Twitter (cf. Table 6.3), we now study how often software developers communicate with each other on Twitter and how different it is from their communication with each other on software project mailing list. Therefore, in the next section we investigate the social interactions of software developers with each other across different communication channels.

6.2 Social Behavior of Developers Across Different Communication Channels

As mentioned in Chapter 3, software developers use a variety of software tools with underlying repositories in order to interact with each other or to solve software-related problems. Apart from mining these software repositories to: predict future software defects [Nagappan et al., 2006; Zimmermann et al., 2007], modeling expertise of software developers [Matter et al., 2009], identifying communication and coordination issues among software developers [Herbsleb and Mockus, 2003; Herbsleb et al., 2001] etc., much research had been carried out in the past to analyze the social network structure and social behavior of software developers by extracting rich information from these repositories [Crowston and Howison, 2003; Long and Siau, 2007; Wiggins et al., 2008]. Fur-
ther, the social behavior of users had been studied in depth in the past on different communication channels separately.

On the usage of social media channels by software developers, [Black et al., 2010] used an online survey to investigate whether social media channels are being used by software developers and whether their use have been successful. However, to the best of our knowledge, no research work has been done so far on the comparison and analysis of the social behavior of software developers across different communication channels. There is no research work available that analyzes the behavior of software developers interaction with each other on the software repositories (e.g., mailing list, bug repositories etc.) and social media channels (e.g., Twitter, Facebook etc.). It is intriguing to see if and how often software developers use Twitter as a sideline communication channel to interact with each other. Moreover, we want to know how often software developers communicate with each other on Twitter and whether there is a correlation between software developers communication with each other across different communication channels. This will further help us to investigate if the social structure of software developer communities on social media channels is similar to those found in other communication channels (e.g., mailing list) [Bougie et al., 2011]. This motivates us to study the social communication patterns of software developers across different communication channels and publishes our findings in [Iqbal et al., 2013].

Among different social media channels available to date, we choose Twitter as a social media channel for this study. Our initial investigation reveals that software developers that are contributing to open source software projects, also use and communicate with each other on the social media channels. For example, Figure 6.5a shows the social network structure (derived from the communication happened on the mailing list) of software developers of an Apache software project. Among them, some software developers are also found on Twitter. Thus, we also derive the social network structure of software developers on Twitter based on their tweets (e.g., mentioning other software developers in tweets), which is shown in Figure 6.5b. We have removed the labels from nodes (cf. Figure 6.5a and Figure 6.5b) in order to keep the privacy of software developers.

![Figure 6.5: Social network structure of software developers in different communication channels of an Apache software project. The nodes in both figures represent same software developers communicating with each other in different communication channels.](image)

---

5 We use the term “communication channels” to refer to software project related communication channels (e.g., mailing list, bug repository, discussion forums etc.) and online social media channels (e.g., Facebook, Twitter etc.).

6 Software developers who have commit rights on the Apache software project.
The different social network structures of software developers (cf. Figure 6.5a and Figure 6.5b) that are contributing to the same software project provides us the basis to investigate the social behavior of software developers in different communication channels. In particular, we investigate if software developers use Twitter as another medium of communication in contrast to the traditional medium of communication (mailing lists, bug repositories, discussion forums etc.). This lays down the foundation to study the social behavior of software developers with each other across different communication channels. In this study, we do not take into account what software developers are discussing on Twitter but instead focus on the quantitative analysis and compute the communication happened between software developers on Twitter in a given period of time and compare it with their communication happened on software repositories for the same period of time. As such, we design an experiment and aims to provide answer to the following research question:

q: Is there any correlation between software developers communication with each other on different communication channels?

We investigate if the software developers communication with each other in different communication channels are correlated or not. We hypothesize that if two software developers are communicating with each other on the software repositories in a given period of time then it is likely that they are also communicating with each other on social media channels during the same time period.

In order to find answer to the above stated question, we compute the social interactions among software developers in different communication channels over the period of time, which is discussed next.

6.2.1 Social Relation Computation Approach

In order to compute the social interactions of a software developer with other fellow software developers on Twitter, we first manually checked if software developers exists on Twitter and using the Twitter account frequently. We found some software developers who does exist on the Twitter platform but tweeted very little (≈10-20 tweets only). We ignored such software developers in this study.

We construct the social network structure based on the mailing list archives by using the reply structure of the email threads for direct communication among software developers. This approach defines a link as the interaction between the poster of actual email and the replier to the poster email. For example, Listing 6.3 indicates that the email is a reply of another email, which is reflected by sioc:reply_of property (see line#12 in Listing 6.3). Hence, we can easily query the poster of email 4C5D409A.9060901 (see line#12 in Listing 6.3) in order to create a social link between both software developers.

In the case of bug tracking repository, we define a link based on the comment posted by a software developer on a particular bug report and the previous commenters on the same bug report.

We construct the social network structure based on the Twitter dataset by exploiting the common practice of using well defined markup in a tweet, i.e., “@” followed by a user identifier. This approach defines a link as the interaction between the software developer who posted the tweet and the software developer mentioned in the tweet. For example, Listing 6.2 tells that the tweet mentions 2 software developers, which is reflected through sioc:mentions property (see line#12-13 in Listing 6.2). Hence, we create a social link between software developers ulander, MaestroDev...
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix siocct: <http://rdfs.org/sioc/types#> .
@prefix email: <http://simile.mit.edu/2005/06/ontologies/email#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .
@prefix : <http://example.org/maven/email2rdf/> .

:A9DB451E-4F0F-435F-8E13-9F4D86996BA4 a siocct:MailMessage ;
e-mail:from <http://example.org/maven/author/brettporter> ;
dct:subject "Re: guice & memory usage was:" ;
e-mail:body "There’s been little to no feedback on beta-2 so ..." .
sioc:reply_of :4C5D409A.9060901 ;
dct:date "2010-08-18T22:43:37+02:00"
.
<http://example.org/maven/author/brettporter> a foaf:Person ;
foaf:name "Brett Porter" ;
foaf:mbox <mailto:brettporter@example.org> .
...

Listing 6.3: RDF representation of an email which is a reply to another email.

and brettporter. Thus, social interactions between software developers on Twitter is computed by querying the Twitter dataset using sioc:mentions property (see query in Listing B.9).

Social interactions between any two software developers are computed only if both software developers have communicated directly to each other on the mailing list and Twitter. Periods without any communication are common as software developers may still contribute to the same software project even if they don’t communicate for some days. To tackle this issue, communication between software developers are captured on monthly basis where each month value represents the number of times both software developers communicated directly to each other. Capturing communication information on monthly basis provides significant amount of data for every pair of software developers in order to analyze their social communication patterns over the period of time.

For every pair of software developers, it is likely that they do not start their communication with each other on mailing list and Twitter at the same day. Therefore, we calculate the initial timestamps where both software developers communicated to each other on mailing list and Twitter. The later date is then used as the starting timestamp to compute the social interactions between both software developers. For example, earliest communication happened between 2 software developers on the mailing list is 2008-05-25 and on Twitter it is 2010-03-15. Thus, we consider 2010-03-15 as the starting timestamp value and compute monthly social interactions between both software developers on mailing list and Twitter over the period of time.

6.2.2 Evaluation

Before we discuss the results of our evaluation, we describe the software projects selected for evaluation. We gather data from software repositories of 10 Apache software projects listed in Table 6.3.

We select data from the beginning of each Apache software project to date. The primary source of communication among software developers in Apache software projects is through mailing lists. Most Apache software projects have at-least 3 different mailing lists: user, dev and commits but some software projects have more than 3 mailing lists (e.g., announcements, notifications etc.). For our study, we consider only the dev mailing list archives of each Apache software project. The reason is, software developers communicate often with each other on the dev mailing list rather
than any other mailing list. From the source control repository datasets of each Apache software project, we retrieve a list of software developers who have made commits to the software project. Later, we manually checked if these software developers exist on Twitter and are using the Twitter account frequently. Table 6.3 shows the total number of software developers who have made commits to the source control repository of each Apache software project and among them, those who are also found on Twitter.

Based on the methodology described in previous section, we found 107 distinct pairs of software developers who have communicated with each other on the mailing list and Twitter. In the specific case of Apache OrBizz software project, we do not find even a single pair of software developers who had communicated with each other on mailing list and Twitter. For each pair of software developers, we compute on monthly basis the number of times both interacted directly with each other on the mailing list (see query in Listing B.10) and Twitter (see query in Listing B.9). As an example, we show a pair of software developers who communicated with each other over the period of time in Figure 6.6. The figure shows that communication pattern among both software developers is different throughout the time period under consideration. For example, in 2009-08-12 both software developers communicated directly with each other 7 times on the mailing list in contrast to 6 times communication on Twitter for the same month. Further, we see certain months where both software developers didn’t communicate at all (e.g., 2011-09-12) and in certain months they appear to communicate on only one communication channel (e.g., 2011-03-12).

---

**Figure 6.6:** Social communication between 2 software developers on Twitter and mailing list in different periods of time.
Given the social dynamics of software developers in different communication channels (cf. Figure 6.6), we evaluate if there is a correlation between communication patterns of both software developers on mailing list and Twitter, through measuring Pearson correlation. The Pearson correlation test based on the communication data between both software developers yield a correlation value, \( r=0.447 \). The \( r \) value indicates that the social communication between both software developers on different communication channels is significant and there is a positive correlation between both software developers communication with each other on different communication channels.

In order to find out if the communication among software developers on different communication channels is directly proportional, we aggregate the monthly communications happened on the mailing list and Twitter for every pair of software developers and plot the resulting graph in Figure 6.7. The graph shows that for majority of software developer pairs, the communication on mailing list and Twitter is not directly proportional.

The highest correlation value we found is for a software developer pair with the value (193,105) where 193 indicates the interaction counts on mailing list and 105 on Twitter. We calculate the Pearson correlation for that particular pair based on their monthly communication data, which results in \( r=0.522 \). Similarly, the lowest correlation value we found is for a software developer pair with value (107,1), which results in \( r=-0.040 \). Additionally, we calculate the Pearson correlation for all software developer pairs and compute the mean and median values, which are shown in Table 6.8.
Table 6.8: Mean and Median of correlation values.

<table>
<thead>
<tr>
<th>Software Developer Pairs</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>0.191</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Q: **Is there any correlation between software developers communication with each other on different communication channels?**

Based on the results in Figure 6.7 and Table 6.8, we can say that the correlation between communication of software developers on Twitter and mailing list is not strong. One potential reason could be the 140 characters limitation on Twitter, which keeps software developers to communicate more through traditional communication channels (i.e., mailing list, IRC channels, forums etc.). Other reason may be the usage of Twitter to communicate non-work related activities. However, it is quite interesting to observe how software developers communicate with other fellow software developers in different communication channels. During our evaluation, we observed significant number of software developers who had communicated less on Twitter but their communication on software project repositories were strong.

As a next step for future work, we will consider social communication graph of a software developer with fellow software developers as well as non-software developers on Twitter. Based on that, we will be able to understand if software developers communication patterns on Twitter is low in general or if they communicate less with fellow software developers only.

### 6.3 Conclusion

In this chapter, we have identified social media channels as a platform that is used by the open source software community and software developers to disseminate software project related information to a wider audience. Further, we highlighted the need to integrate software repositories and social media channels (interlinking software project related tweets/posts/hashtags, software developer IDs, software project IDs etc.) in order to get an integrated view on the software project. We studied the usage of software project related hashtags by software developers on Twitter and found that software developers do use hashtags to communicate about the software projects they are working on. However, the ratio of usage of software project related hashtags vary for each Apache software project we considered in the study (cf. Table 6.7). Moreover, we found that software developers of some Apache software projects used software project related hashtags more often than any other hashtag (cf. Table 6.6), which tells that certain software projects are frequently discussed on Twitter by software developers.

We have also motivated and introduced a new dimension to the analysis of social dynamics of software developers by taking into account social media channels. The usage of social media channels is becoming popular among open source software community and software developers to disseminate software project related information to a wider audience. This motivated us to investigate if the communicational behavior of software developers is same across different communication channels or not. Our initial investigation based on the data of 10 different Apache software projects showed that there is a low correlation between software developers communication with each other across different communication channels. In addition to that, our results...
showed that the social interactions between software developers on Twitter is comparatively low than the traditional communication channels (i.e., mailing lists, bug repositories etc.).

The work presented in this chapter lays down the idea of taking into account all possible social media channels that software developers could possibly use to communicate with each other. Based on that, researchers will be able to measure and compare the hierarchy and centralization of software developers in different communication channels in contrast to previous studies where researchers had been using only software repositories [Crowston and Howison, 2005; Wiggins et al., 2008]. Furthermore, integrating social messages/posts to software project related artifacts will enable us to analyze the impact of end-user’s response on the success/failure of an open source software project.
Research in software development has shown that identification and management of work-related dependencies is a fundamental challenge, particularly for software development teams who are geographically distributed [Cataldo et al., 2007; Grinter et al., 1999; Herbsleb and Mockus, 2003]. Further, it has been recognized long ago that breakdown in coordination and communication efforts constitute a major issue in software development [Curtis et al., 1988], mainly due to the large number of evolving dependencies: dependencies among software developers due to interdependent tasks and dependencies among different software artifacts [Trainer et al., 2005]. Different tools and methodologies have been developed by the software engineering community to overcome dependency issues. Examples of such tools are mailing lists, bug repositories, configuration management etc. Minimizing dependencies among software developers reduces the required communication and coordination needs [de Souza et al., 2007]. This was recognized by Parnas [Parnas, 1972] and later validated by many empirical studies [Grinter, 2003; Morelli et al., 1995; Sosa et al., 2002].

In this chapter, we focus on identifying the social dependencies and social relation among software developers over the period of time by exploiting our interlinked dataset. We investigate the socially dependent and socially related software developer pairs as well as the social dependency and social interaction of software developer pairs over the period of time. The remainder of this chapter is structured as follows:

• In Section 7.1, we investigate how socially dependent and socially related software developer pairs changes over time. We compute the socially dependent and socially related software developer pairs on monthly basis and plot it in the form of charts.

• After identifying the socially dependent and socially related software developer pairs over the period of time, we select some software developer pairs and investigate their social dependency and social relation with each other over time in Section 7.2.

• In Section 7.3, we investigate if a software developer communicates with socially dependent software developers and compute the correlation between social dependency and social interaction of a software developer with other software developers.

• Finally, we conclude in Section 7.4.

7.1 IDENTIFYING SOCIAL DEPENDENCIES AND SOCIAL INTERACTIONS

Software developers often work on dependent piece of source code and hence more likely engage in communication or coordination prior to changes in the source code. Using our interlinked
datasets\textsuperscript{1}, we perform preliminary analysis on the social dependency and social relation among software developers of some selected Apache projects in order to investigate how the dependencies and communication among software developers changes over the period of time.

7.1.1 Identifying Socially Dependent Software Developer Pairs

Creating social dependencies among software developers require collecting source code dependency information and retrieving authorship information from source control repository in order to identify software developers associated with the dependent source code. However, in this section we are interested to investigate how social dependencies among software developers changes over the period of time. As the software project evolves over time, we expect that the social dependencies among software developers also changes. Thus, we investigate how the dependencies among software developers changes as the project continue to evolve over the period of time. It enables us to understand if the social dependencies among software developers increases, decreases or remains the same as the project matures over time. We want to highlight that we do not consider the call-graph (cf. Section 2.4.1.1) of a software project to identify the dependencies among different source code. The reason is that we are identifying social dependencies among software developers over the period of time and in order to build social dependencies among software developers over time using call-graph, we are required to build various source code snapshots of a software project. Therefore, we follow another approach and construct social dependencies among software developers who worked on the same source code over the period of time. We explain our approach in the following.

To identify social dependencies among software developers for a given period of time, we first query (see query in Listing B.11) the source control repository to retrieve a list of source code files that are committed during that period of time. Quite often a single source code file is committed multiple times on the source control repository by different software developers in a given period of time. Software developers committing source code files on the source control repository are considered to be the authors of those source code files. Hence, all software developers who committed a particular source code file on the source control repository in a given period of time are considered to be socially dependent on each other. Therefore, for each source code file committed, we query (see query in Listing B.12) the source control repository to retrieve a list of software developers who committed that particular source code file. Later, we group all those software developers into socially dependent software developer pairs. For example, if \texttt{JasperException.java} is committed by dev-A, dev-B and dev-C on a source control repository in time period \textit{y}, then socially dependent software developer pairs will be: \{\texttt{dev-A,dev-B}\}, \{\texttt{dev-A,dev-C}\} and \{\texttt{dev-B,dev-C}\}. This means that all three software developers must communicate with each other as they work on the same source code file in a specific time period. We have calculated the socially dependent software developer pairs of Apache OfBiz project on monthly basis and plot the results in the form of a chart shown in Figure 7.1.

In order to keep the privacy of software developers, we only show how many software developer pairs are socially dependent in a given period of time. Figure 7.1 gives an overview on how many software developers worked on similar source code files in a given period of time. For example, the highest peak point when most software developers are socially dependent is the month 49 of

\textsuperscript{1} For the sake of privacy of software developers who worked on multiple projects that are presented in this thesis, we do not provide interlinked datasets for others to explore. However, interested readers can adopt our methodology and generate the interlinked datasets for research and to further explore the datasets.
the software project where 54 distinct software developer pairs are found to be socially dependent on each other. This means that most software developers worked on same source code files during this month comparing to other months of the software project. On the contrary, take month 84 as an example where only 1 software developer pair is found to be socially dependent, which means that only two software developers worked on same source code files during that particular month.

We looked into the number of commits made to the source control repository during these months in order to find out if the number of commits made to the source control repository had any effect on the socially dependent software developers pairs. We found that 545 commits are made in month 49 and only 43 commits are made in month 84 on the source control repository. Based on this observation, we can say that if the number of commits made to the source control repository is high, then there is a higher chance of an increase in social dependencies among software developers during that period of time. We have also calculated the social dependencies among software developers of some selected Apache projects on monthly basis and plot their results in the form of charts that are shown in Section A.1 of Appendix.

7.1.2 Identifying Socially Related Software Developer Pairs

Software developers are considered to be socially related when they communicate directly to each other on a communication channel. Therefore, the social relation among software developers based on the mailing list dataset is constructed by using the reply structure of the email threads.
for direct communications among them. This approach defines a link as the interaction between the poster of actual email and the replier to the poster email.

In order to identify socially related software developer pairs in a given period of time, we first query the mailing list dataset (see query in Listing B.13) to retrieve all those emails that are sent in response to other emails. Later, for each email we query the mailing list dataset (see query in Listing B.14) to retrieve the message-ids for which that particular email is a response. If an email is a reply to another email then the message-id of the actual email can be found in the email header represented by the reference keyword (see line #6 of Listing 3.7). While converting an email to RDF, we have used sioc:reply_of property to represent the message-ids that are associated with the reference keyword (see line #12 of Listing 3.8). Quite often, an email contains multiple message-ids, which means that there are multiple responses to the actual email. Therefore, we retrieve all message-ids that are present in a particular email using the sioc:reply_of triple pattern (see query in Listing B.14). Later, we retrieve the sender information (i.e., software developer’s name) by using those message-ids. By following this approach, we are able to easily identify the name of the actual poster of the email and all those who have replied to the poster email.

As anyone who is subscribed to a mailing list can send an email or reply to an email but we are particularly interested in the social relation among software developers. Therefore, we validate each person’s name against software developer names. We perform this validation by retrieving software developer names using software developer’s subversion (SVN) IDs. Based on this approach, we filter out all those who are not software developers. As a final step, we group software developers into socially related software developer pairs. As the software project evolves over time, we expect that the socially related software developer pairs also changes. Therefore, we are interested to investigate how the socially related software developer pairs changes as the software project evolves over the period of time. We have calculated the socially related software developer pairs of Apache OrBiz project on monthly basis and plot the results in the form of a chart shown in Figure 7.2.

In order to keep the privacy of software developers, we only show how many software developer pairs are socially related in a given period of time. It gives an overview of how many software developer pairs are socially related in a given period of time. For example, the highest peak point when most software developer pairs are found to be socially related is the month 37 of the project, where 63 distinct software developer pairs are found to be interacted with each other. This also means that most software developers communicated directly to each other in that particular month in contrast to other months. On the other hand, only three software developer pairs are found to be socially related during month 83. We have also calculated the socially related software developer pairs of some selected Apache OrBiz projects on monthly basis and plot their results in the form of charts that are shown in Section A.2 of Appendix.

7.1.3 Discussion

As mentioned earlier, technical dependencies among source code creates social dependencies among software developers who are working on technically dependent source code. Hence, the social dependencies foster the needs of communication and coordination among software developers in order to align the work. This is also reflected in the results that are extracted from software repositories of Apache OrBiz project (cf. Figure 7.1 and Figure 7.2). The highest peak points of socially

\footnote{In this chapter, we use the term “software developer” to represent those who have write-access to the source control repository.}
dependent software developer pairs (cf. Figure 7.1) are observed between years \((Y-2,Y-3)\) as well as between \((Y-3,Y-4)\), where a total of 328 software developer pairs are found between \((Y-2,Y-3)\) and 466 software developer pairs are found between years \((Y-3,Y-4)\). We summed up the software developer pairs value of each month in a year to compute the total number of software developer pairs for that particular year. Further, looking into the socially related software developer pairs (cf. Figure 7.2), we found the highest peak points of socially related software developers (509 software developer pairs in \((Y-2,Y-3)\) and 528 in \((Y-3,Y-4)\)) during the same years under consideration.

Given the dynamics of socially dependent and socially related software developer pairs of Apache OfBiz project, we evaluated if there is a correlation between socially dependent and socially related software developer pairs of Apache OfBiz project over the period of time by measuring Pearson correlation. The Pearson correlation yielded a correlation value, \(r=0.68\). The \(r\) value indicates that the increase in socially dependent software developer pairs also increases the socially related software developer pairs and vice versa. Further, we calculated the correlation value for socially dependent and socially related software developer pairs of each Apache project (based on the data used to plot charts shown in Section A.1 and Section A.2 of Appendix) and computed the mean of all correlation values, which resulted in \(r=0.44\). The mean correlation value also indicates a significant value and shows a positive correlation between socially dependent and socially related software developer pairs of all Apache projects (under consideration) for a given period of time.

Software developers can use any medium of communication (e.g., mailing lists, bug repositories, IRC channels etc.) to communicate with each other, however we have calculated and plotted the
results based on communications happened among software developers on the mailing list only. We might achieve high correlation value by taking into account other mediums of communication, which we have left out in this study.

7.2 STUDYING SOCIAL DEPENDENCIES AND INTERACTIONS OF DEVELOPER PAIRS

In the previous section, we investigated the socially dependent and socially related software developer pairs of some selected Apache projects over the period of time. However, we did not investigate if a socially dependent software developer pair in a certain period of time was also socially related during the same time period. In other words, do the software developers who are found socially dependent in a certain month communicate with each other in the same month or not? We want to emphasize that our investigation is based on the quantitative (i.e., number of commits, number of interactions etc.) rather than qualitative analysis.

7.2.1 Social Dependencies of Software Developer Pairs

It is common that software developers work on interdependent source code and hence likely to be socially dependent on each other throughout their development activity on a software project. Therefore, we study how the social dependency among a pair of software developers changes over the period of time. In the previous section, we presented only the number of software developer pairs who are found to be socially dependent in different periods of time. However, in this section we consider some software developer pairs and investigate their social dependencies on each other as they worked together on the software project over the period of time.

As there are large number of software developer pairs identified in each month (cf. Figure 7.1) and it is difficult to present the social dependencies of all software developer pairs over the period of time in the form of chart. Therefore, we consider only the top three software developers who are found to be socially dependent most of the time throughout the software project time period under consideration. We first extract the distinct software developer pairs found in our data and later for each software developer pair, we count how many times each software developer pair appear to be socially dependent. Once the software developer pairs are ranked based on the number of times they appear to be socially dependent, we select the top three software developer pairs and plot their social dependency on each other over the period of time as shown in Figure 7.3.

Figure 7.3 shows that the social dependency factor (y-axis) is continuously changing for software developer pairs over the period of time. The social dependency factor tells how many source code files are authored (i.e., committed on source control repository) by a software developer pair in a certain month. For example, pair 3 committed 39 source code files in month 7 of the software project, which means that both software developers worked together on 39 source code files in different time intervals of month 7. Further, we see that pair 1 and pair 2 worked on similar source code files most of the time duration of the software project, however pair 3 stopped working on similar source code files from year 5 onwards.
7.2 Studying Social Dependencies and Interactions of Developer Pairs

7.2.2 Social Interactions of Software Developer Pairs

Communication and coordination is an important aspect of software development in order to guarantee a smooth flow of work. In Section 7.1.2, we showed only the number of software developer pairs who are found to be socially related in different periods of time. On the contrary, in this section we consider some software developer pairs and investigate their social interactions with each other by extracting direct communications among them on the mailing list over the period of time.

In the previous subsection (cf. Section 7.2.1), we considered three software developer pairs who are found to be socially dependent on each other most of the time throughout the time period under consideration. Therefore, we select the same three software developer pairs, compute their social interactions with each other throughout the time period under consideration and plot the results in Figure 7.4. We chose the same software developer pairs so that we can compare their social dependency factor against their social interaction factor. We hypothesize that if the social dependency between two software developers is higher in a particular time period then both software developers will have higher number of social interactions with each other.

Figure 7.4 shows the diversity in social interactions among software developer pairs as it continuously changing over the period of time. The social interaction factor shows how many times both software developers communicated directly with each other on the mailing list. This is derived by extracting all emails which are sent by these software developers and later identifying if the email...
is a reply to an email posted by other software developer. For example, pair 2 communicated 250 times with each other in month 72 of the software project. Further, we see that all three software developer pairs were communicating actively with each other throughout the time period of the software project.

7.2.3 Discussion

In this section, we explored how the social dependencies and social interactions among software developer pairs changes over time by taking into account three software developer pairs of Apache OfBiz project. The results showed that the social dependencies and social interactions among software developers is highly dynamic in nature and there is no pattern exists as such. However, we found that in certain months, software developer pairs are found to be socially dependent but there was no social interaction among them on the mailing list. For example, social dependencies of pair 1 started to evolve from month 5 but their social interactions started to evolve from month 10 on the mailing list. It is possible that both software developers have communicated earlier through other communication channels (e.g., IRC channels), which we have not considered in this study.

As prior research revealed that coordination among software developers is necessary before modifying interdependent source code, we have found similar case in our study when a software developer pair started to communicate with each other before their social dependencies started.
to evolve. For example, pair 2 started to communicate directly with each other on the mailing list from month 4, however their social dependencies started to evolve from month 17 onwards. Communication among software developers is common irrespective of any social dependencies, which exists among them. For example, there was no social dependency among pair 3 from year 5 onwards, however they still communicated with each other on the mailing list for year 6 and year 7. We have computed the social dependencies and social interactions among top three software developer pairs of other Apache projects and plot their results in the form of charts that are shown in Section A.3 of Appendix.

In this section, we have explored how the social dependencies among software developer pairs evolved and compared it with their social interactions on the mailing list. However, we believe that considering all possible communication channels (mailing lists, bug repositories, discussion forums, IRC channels etc.) might provide further insights into the social dynamics aspect of software developers.

7.3 Correlation between Social Dependency and Interaction of Developers

In the previous sections, we studied the socially dependent and socially related software developer pairs of Apache projects over the period of time. We presented the quantitative results of socially related and socially dependent software developer pairs identified over a period of time (cf. Figure 7.1 and Figure 7.2). Further, we considered three software developer pairs of each Apache project and studied how their social dependency and social interaction changes over the period of time. We again presented the quantitative results for social dependencies and social interactions of software developer pairs over the period of time (cf. Figure 7.3 and Figure 7.4). However, we did not investigate if a software developer who was found to be socially dependent on other software developers, actually interacted with them during that time period or not. We investigate this particular aspect in this section.

In order to calculate the social dependencies and social interactions of a software developer with other fellow software developers over the period of time, we first retrieved a list of software developers by executing a SPARQL query (see query in Listing B.15) on the source control repository dataset. Later, for each software developer we computed his/her first and last commit date on the source control repository. We used the first and last commit date to compute the social dependencies and social interactions of that particular software developer with other software developers of the software project.

In order to compute the social dependencies of a software developer with other software developers, we first retrieve a list of source code files, which are committed by a software developer in a given period of time (see query in Listing B.16). Later, for each source code file we execute SPARQL queries (see query in Listing B.12) to retrieve a list of software developers who have committed the same source code file during that specific period of time. Finally, we consider all those software developers who have worked on the same source code files to be socially dependent with the software developer under consideration. In order to compute the social interactions of a software developer with other software developers, we consider the same time period (first and last commit date), so that we compute the social interactions of a software developer on same time intervals which we use for computing social dependencies. Before computing the social interactions of a software developer with others, we query our interlinked dataset to retrieve a list of IDs that are associated with that particular software developer (see query in Listing B.17). It
is possible that a software developer has used multiple IDs (cf. Section 4.1) to communicate with other fellow software developers, which is the reason that we have considered multiple IDs of a software developer while computing his/her social interactions with other fellow software developers during that specific period of time. We have plotted the results that shows the number of software developers who are found to be socially dependent and socially related with a software developer in Figure 7.5.

![Figure 7.5: Social dependency and social relation of a software developer with other software developers over the period of time.](image)

The results (cf. Figure 7.5) shows that the software developer has higher number of socially related software developers in contrast to the socially dependent software developers throughout the time period under consideration. However, it still do not tell us if the socially dependent software developers also exists among the socially related software developers. Therefore, we use our socially dependent and socially related data that we have computed for a software developer, extract the software developer names who are socially dependent and matched them against the software developer names who are socially related during the same time period. This way we can find out how many software developers are socially dependent in a given time period and how many of them communicated with him/her during the same time period. We plotted the results based on our findings for a single software developer that we considered earlier (cf. Figure 7.5) in the form of a chart shown in Figure 7.6.

The figure shows the number of software developers who are found to be socially dependent and those who actually interacted with him/her during the same time period. For example, during time interval 2001-11-26 there are 6 software developers who are found to be socially dependent
and 28 software developers are found to be interacted with each other (cf. Figure 7.5). Among those 28 software developers, we found the existence of 6 software developers with whom the software developer have social dependencies (cf. Figure 7.6). Hence, the software developer interacted with all socially dependent software developers during time interval 2001-11-26. On the contrary, we have found some time intervals, where the software developer do not communicate with all software developers with whom he/she share social dependencies. An example of such is the time interval 2001-10-26. There is a possibility that the software developer communicated with them on other communication channel (e.g., IRC Channel), which we do not consider in this study.

7.3.1 Correlation

Given the dynamics of social dependencies and social interactions of a software developer with other fellow software developers over the period of time, we evaluate if there is a correlation between the socially dependent and socially related software developers for a particular software developer (cf. Figure 7.5) by measuring Pearson correlation. The Pearson correlation yielded a value, \( r = 0.67 \). The correlation value indicates that the increase in the social dependency of a software developer also increases his/her social interactions with socially dependent software developers. Similarly, we calculate the correlation value of all software developers of each Apache project and compute the mean of all correlation values, which resulted in \( r = 0.44 \). The mean correlation value

Figure 7.6: Number of socially dependent software developers and those who actually had social interactions with him/her during the same time period.
also indicates a significant value and shows a positive correlation between social dependencies and social interactions of a software developer with others in a given period of time.

Moreover, we computed the average percentage of a software developer interaction with fellow software developers with whom he/she share social dependencies (based on Figure 7.6), which resulted in 84.86%. Similarly, we computed the mean of all software developers of each APACHE project and computed the mean of all percentages, which resulted in 50.11%. The mean value of all software developers turned out to be quite low. One reason behind this might be the usage of other communication channels (i.e., IRC channels), which we have not considered for this study.

### 7.4 Conclusion

The main aim of the work presented in this chapter is to exploit our interlinked datasets to provide a glance on the evolving social interactions and dependencies among software developers over the period of time. Our first evaluation aimed at identifying the socially dependent and socially related software developer pairs over the period of time, depicting an overall picture of the evolving dependencies among software developers working on a software project. We found that there is a correlation between evolving social dependencies and social relations among software developer pairs over the period of time. In our second evaluation, we studied the social dependencies and social relations of some software developer pairs over the period of time. We found dynamics in the social dependencies and social interactions of software developers with each other over the period of time. Moreover, we found certain cases in our results, where a software developer pair had a social dependency but no social interactions in a given period of time and vice versa. We believe that such a study is useful to analyze the patterns of evolving social dependencies and communication behavior of a software developer with fellow software developers over the period of time. Further, it can help project managers and lead software developers to get useful insights into how the dependencies among software developers change over time and if such dependencies make software developers to communicate with each other or not. We further extended our evaluation and investigated if there are social interactions among software developers who are found to be socially dependent in a given period of time. We calculated the correlation based on social dependencies and social interactions in a given period of time, which turned out to be positive. Hence, this shows that an increase in social dependency also increases the social interactions among software developers.
Software development does not revolve around a piece of source code but also around large volumes of software development related information that exists in different software repositories and are used by software developers to manage their development tasks. With the advent of collaborative platforms to allow collaborative software development on the Web, we have seen a tremendous growth in the availability and usage of different code forges promoting and adopting OSS development. Moreover, there also exists statistical services on the Web that provides detailed analytical information about software projects and software developers. Last but not least, we have witnessed extensive usage of social media platforms by the software communities and software developers to disseminate information to a wider audience. Hence, we can say that information related to software projects and software developers are distributed on the Web in heterogeneous software repositories (i.e., code forges, social media channels, statistical services etc.).

The presented thesis was motivated by the observation that there is a lack of integration among software repositories based on similar or related software artifacts. Thus, we have proposed to exploit a standard approach that allows an easy identification and integration of software project related information across heterogeneous software repositories.

8.1 Contributions

Contribution to H1

*Integrating software project and software developer related information across OSS repositories and social media channels enable a comprehensive overview on the development of an OSS project.*

We have shown that integrating related information across different software repositories enable a comprehensive overview on the development history of software projects and software developers. We have proposed a Linked Data approach and argued that Linked Data is a better way to achieve integration of information across different software repositories on the Web. We have explained with examples how to represent information contained inside different software repositories (source control repository, bug repository, mailing list etc.) hosting a software project, statistical services such as Ohloh, code forges database dumps provided by FLOSSMOLE and tweets posted by software developers on Twitter in RDF. Later, we demonstrated how the integration can be achieved between different software repositories by establishing RDF links across different RDF resources.

Once we have the interlinked RDF data space, we presented SPARQL queries to demonstrate that it allows software stakeholders to not only query software artifacts that are hidden deep inside the software repositories hosting a software project but also enables them to query statistical information about software projects and software developers (from Ohloh). Moreover, we have shown that interlinking software developer IDs enable software stakeholders to keep track of the information relevant to software developers on different code forges, statistical services (i.e., Ohloh) and their development activities across different software repositories of a software project as well
as across multiple software projects. In addition to that, we investigated the use of social media channels (Twitter in particular) by software developers and based on our findings argued that software developer and software project related information that are shared on the social media channels can also be integrated to enable a comprehensive overview from the development and social perspectives.

CONTRIBUTION TO H2

*We can efficiently query integrated software data and deliver related information about a particular software development task as well as software developers activities within and across different OSS projects automatically and efficiently.*

With “Linked Data Driven Software Development (LD2SD),” we have introduced a Linked Data based, light-weight framework that allows software stakeholders to exploit the interlinked information that are contained in various software repositories hosting a software project. We have developed an interface on top of LD2SD that software stakeholders can use either as a plugin (LD2SD plugin) in their development environment (e.g., Eclipse IDE) or as a Web based application to retrieve relevant information about a software artifact (e.g., Java Class, Java Package, Bug, Source Control Commit log etc.) from different software repositories. With respect to its effectiveness, we investigated the traditional approach (i.e., manual) that software developers usually follow to find relevant information about a particular task from various software repositories against our automated approach by conducting an end-user evaluation. The results of the study showed that the users who participated in the evaluation spent less time to perform certain pre-defined tasks using our automated approach in contrast to the manual approach for searching related information due to the lack of integration among software repositories.

In addition to that, we have supported with evidence that software developers often contribute to multiple software projects. Therefore, the development activity of software developers are distributed not only within different software repositories hosting a software project but also across multiple software projects. Moreover, software developers often use different and sometimes multiple identities while interacting with these software repositories. Hence, we have shown that interlinking software developer IDs help software stakeholders in keeping track of the development activities of a software developer within a software project as well as across multiple software projects.

CONTRIBUTION TO H3

*We can efficiently query integrated software data to assess various technical and non-technical factors, such as:

- software developers development ratio;
- software developers development activities.*

We showed in our experiments that software stakeholders can efficiently query and exploit the interlinked information of software repositories, code forges and statistical services in order to assess various technical and non-technical factors. Through interlinking software developer IDs across different open source repositories (software repositories, code forges, statistical services), we demonstrated that we were able to trace the development activity of software developers across different software projects, their involvement in multiple software projects on different code forges as well as their development statistics on different software projects that are made available by Ohloh.
We can efficiently query integrated software data to assess social factors, such as:
- social dependencies and relations among software developers;
- software developers communication behavior across different communication channels.

We have shown that we were able to exploit interlinked dataset in order to investigate the evolving social dependencies and social interactions of software developers with each other over the period of time. Our results showed that there is a positive correlation between evolving social dependencies and social relations among software developer pairs over the period of time.

We extended our study on the communication aspects of software developers and introduced a new dimension to the analysis of social aspects of software developers by taking into account social media channels. We studied the usage of social media channels by software developers through harvesting their project related activities. Our initial investigation revealed that software developers do use other communication channels (e.g., Twitter) to communicate with fellow software developers. Thus, our initial study laid down the foundations to consider all communication channels that software developers could possibly use in order to measure the hierarchy and centralization of software developers in different communication channels in contrast to previous studies where researchers had used only software project related software repositories (i.e., mailing lists, bug repositories etc.).

8.2 LESSONS LEARNT

In this section, we discuss some of the constraints/limitations that are encountered in our research and the related lessons learnt.

CONTINUOUS INTEGRATION OF SOFTWARE ARTIFACTS

We have presented preliminary evaluations to study the effectiveness of applying Linked Data approach for integrating different software repositories. We have shown with exemplary SPARQL queries what can be achieved once the information is integrated, however much more can be done along these lines. For example, we considered data dumps of software repositories to transform information about software artifacts into RDF datasets and later integrate them to each other. However in a software development process, software artifacts are continuously changing or emerging. Therefore, event listeners are required to be developed that listens to events (i.e., changes happening in a software repository) occurred in different software repositories, extract the new or updated software artifacts and integrate it to other relevant software artifacts on continuous basis. This is particularly important to provide an up-to-date information to software stakeholders in order to help them in their day to day software development tasks.

INTEGRATING SOFTWARE ARTIFACTS AT WEB SCALE

We have motivated and proposed integration of code forges based on metadata, similar software projects and software developers by taking into account code forges database dumps that are provided by FLOSSMole community. Due to the limited information contained inside database dumps, we provided some preliminary results on modeling code forges metadata and integrating a small subset of similar software projects and software developers across different code forges. However, we believe that a Web scale integration of different code forges based on similar metadata, software projects (along with its underlying software repositories) and software developers can be
achieved in order to develop an OSS ecosystem. This can be then exploited by researchers to have a full picture of the OSS world and to study various aspects of the OSS communities (e.g., trends, technologies, success/failure of software projects etc.).

**SOCIAL MEDIA CHANNELS**

With respect to social media channels, we identified that software communities do adopt an identity on social media channels in order to disseminate information to a wider audience. We considered only Twitter (a microblogging website) as a case study in this thesis to motivate the integration of social media channels with software project repositories. However, other potential social media channels (e.g., social networking websites, blogs and Q&A websites) must also be taken into account that contains useful information relevant to software projects and software developers.

We believe that a Linked Data approach for integrating software project and software developer related information across different open source software repositories and social media channels make a significant step towards adopting Linked Data technologies by software stakeholders. Given the interoperability and integration issues across different software repositories, we believe that the next generation development tools will adopt Linked Data technologies to offer a seamless integration across different software repositories (see Section 8.3.1). This thesis is the first step in this direction.

**8.3 FUTURE DIRECTION**

In this thesis, we have demonstrated how we apply Linked Data to various software repositories in order to enable software stakeholders have a comprehensive overview on the information for their specific needs. We now present briefly what we believe to be important future direction one can take from here.

**8.3.1 Standardised Initiatives**

In the past couple of years, we have seen good examples of standardization initiatives where important organizations within open communities stress on the advantages of easier integrations of software artifacts and data interoperability. The most important and relevant initiatives that are currently under development are the Open Services for Lifecycle Collaboration (OSLC) hosted by OASIS\(^1\) and Linked Data Platform (LDP) promoted by World Wide Web Consortium (W3C). In the following, we briefly describe them and potential work that can be taken into account as a future work of this thesis.

**8.3.1.1 Open Services for Lifecycle Collaboration (OSLC)**

The OASIS Open Services for Lifecycle Collaboration (OSLC) aims at building an open community for developing specifications for integrating software artifacts using well established Internet and Linked Data standards in order to enable sharing of data among software lifecycle tools. The OSLC specifications are build on top of RDF, Linked Data and REST, hence enabling integration at software artifacts level. In particular, OSLC aims at: 1) defining ontologies in order to provide a

---

\(^1\) [http://www.oasis-oslc.org/](http://www.oasis-oslc.org/)
common understanding of software artifacts; 2) exposing and sharing data using the Linked Data principles; and 3) inviting industry experts to explore common issues in software development and integration.

Given that this initiative is backed up by leading software industry experts, therefore we believe it is important to explore how it evolves and can be applied/extended to the work proposed in this thesis. Moreover, vocabularies that are chosen and used in this thesis in order to represent software artifacts in RDF can be aligned with the vocabularies that are produced by OSLC as well as incorporating other types of software artifacts that are not considered in this thesis.

8.3.1.2 Linked Data Platform (LDP)

The W3C Linked Data Platform (LDP) specification defines a set of best practices and approach for a read-write Linked Data architecture. It is solely based on HTTP access to resources that are available on the Web and describe the state of resources using RDF data model. The Linked Data Platform protocol extends the HTTP protocol and specifies how Web applications can find resources, publish new resources, edit and delete existing resources from servers that expose their resources as Linked Data. In particular, Linked Data Platform protocol allows building interoperable read-write Linked Data applications.

In this thesis, we demonstrated what Linked Data is capable of and how it can be used to integrate heterogeneous software repositories on the Web. We showed up-lifting of data from different software repositories into RDF and its integration. However, we do not present if and how these Linked Data resources can be updated and pushed back to the legacy software repositories. Therefore, we believe that Linked Data Platform can play a key role in this direction and is an important area to be explored by others.

8.3.2 Software Development in the Cloud

With the emergence of cloud computing infrastructure, small and large corporations alike have started to move towards hosting their data, software applications, operational communication networks etc., on the large-scale server farms. Time and cost benefits are the leading motivating factors and measures for any business accessing tools and services via the cloud [Willie, 2011]. Popular cloud computing services fit into one of the following categories: **Infrastructure, Platform or Software. Software as a Service (SaaS)** are offered by many service providers, including online project management applications, customer relation management, online office applications and many more. Following the successful adoption of cloud services by the enterprises, different cloud service providers started to offer data access, software, hardware and data storage services. Among them, few efforts takes “as a service” to the software development field where we get **Development as a service (DaaS)**. DaaS is a suite of tools that allows to use traditional development practices for creating on-demand applications. DaaS expands the cloud computing development process to encompass external tools such as integrated development environments, source control systems and collaborative tools to facilitate development and deployment with the aim to facilitate and manage software tools and infrastructure for an enterprise, especially with development teams geographically distributed. By leveraging a cloud platform, an enterprise

---

2 https://dvcs.w3.org/hg/ldpwg/raw-file/default/ldp-primer/ldp-primer.html
3 http://en.wikipedia.org/wiki/Software_as_a_service
4 http://wiki.developerforce.com/page/An_Introduction_to_Metadata_and_Development_as_a_Service
can start using software tools instantly, cost effectively and without managing any development infrastructure [Willie, 2011]. According to [Armbrust et al., 2010], an enterprise can utilize cloud development infrastructure to:

1. Scale on demand.
2. Spend more resources on time-to-market by relieving resources from infrastructure management responsibilities.
3. Achieve higher cost efficiencies.
4. Configure development infrastructure in minutes rather than spending whole day on an installed infrastructure.

Recently, service providers have started to offer browser-based development environment, allowing to edit, test, debug, deploy and manage applications using the browser without the need of installing or configuring any tool or IDE on a local machine. It enables software developers to harness the cloud computing power: just as Platform-as-a-Service (PaaS) enables an enterprise to run applications in the hosted platforms, DaaS provides a new software development stack giving software developers the power to write, deploy and manage software applications in the cloud as depicted in Figure 8.1.

![Figure 8.1: Cloud-based software development stack.](image)

The cloud-based software development stack typically comprises the following components (exemplary coverage overlaid in Figure 8.1):

1. **Editor**—A browser-based code editor, allowing to edit, debug and test source code, for example, Cloud9 IDE.
2. **Deployment**—Deployment of an end-user application in the cloud through a PaaS provider, such as Heroku.

---

5 [http://en.wikipedia.org/wiki/Platform_as_a_service](http://en.wikipedia.org/wiki/Platform_as_a_service)
6 Kudos to Mike Amundsen for coining the term "cloud-stack" programming in his blog post available via: [http://www.amundsen.com/blog/archives/1116](http://www.amundsen.com/blog/archives/1116)
7 [https://c9.io/](https://c9.io/)
8 [https://www.heroku.com/](https://www.heroku.com/)
3. **Data Store**—A cloud-based data storage system (RDBMS or NoSQL) used by an application to store and query application data, for example CouchDB\(^9\).

4. **Source and Project Management Controls**—Managing and sharing software repositories through cloud-based collaborative software development architecture, such as TeamForge\(^10\).

### 8.3.2.1 Towards a Next Generation of Cloud Software Development

In traditional in-house software development, we deal with different software repositories that surround a particular software project. These software repositories are necessary to maintain and execute a software project in a structured way. Based on the emerging trends of cloud-based infrastructures to support software development and different browser-based development tools/frameworks and IDEs that are available to date [Iqbal et al., 2012b], we foresee the development of software projects in the cloud as depicted in Figure 8.2.

![Figure 8.2: Continuous integration and deployment of software applications in the cloud.](image-url)

**Challenges and Opportunities**

In the following we discuss challenges or areas of improvement, which can foster the adaption of cloud-based software development:

1. **Dealing with the agile software development life-cycle.** In agile software development, immediate feedback is essential and it is generated through frequent commits, builds, testing and continuous integration. As the cloud-based development infrastructure offers support for these software tools as a single integrated platform, it is worth investigating if the online


\(^10\) [http://www.collab.net/products/teamforge](http://www.collab.net/products/teamforge)
browser-based IDEs (i.e., code editors) add more efficiency, support and transparency than developing the code in a desktop-based environment. Currently, cloud-based software development tools offer integration with desktop-based development environments but such type of integration is missing with browser-based IDEs. In order to fully support agile software development in the cloud, we need to support integration between different cloud-based development tools and infrastructures.

2. **Automation of tasks.** Cloud-based infrastructure makes it straight-forward to export and share data from software tools (i.e., coding, testing, build logs, integration tests etc.). Real time capturing of data from these tools will enable enterprises as well as software developers to measure performance, tracking activities of a software developer and monitoring the overall progress on the software project. In order to achieve this, service providers must provide plugins/tools which can publish and consume data from different cloud-based development tools.

3. **Integration with code forges.** Hundreds of thousands of software projects are hosted on different code forges. Only some browser-based IDEs support integration with code forges (e.g., Cloud9 supports GitHub integration). Integration with code forges will enable software developers to easily work on software projects that are already hosted on code forges, using browser-based IDE. As code forges hosts a variety of project management tools (i.e., bug tracking systems, mailing lists, discussion forums), it will further assist developers to reuse the existing project management tools hosted on these code forges. For example, a bug report could be fixed by modifying the source code, compiling, building and testing it in a browser-based IDE and later pushing the source control commit changes back to the code forge along with changing the status of the bug report on code forge.

4. **Bridging the “off-line gap”.** Browser-based IDEs should allow software developers to import or export software projects directly from their desktop IDEs to the cloud. A software developer should be able to code on his local machine and a single push button would allow him to push the changes to the cloud so that other team members can access or review the code changes using browser-based IDE. In the same direction, the question has to be answered how (local or remote) backup are supported.

**Requirements**

Based on the challenges and opportunities outlined above, we have derived a set of requirements we consider pivotal for the next generation of cloud-based software development tools.

1. The integration between PaaS and cloud-based development tools is essential; the interfaces should be standardised, both on the data format level (for example, JSON, RDF) as well as on the protocol level (RESTful).

2. Cloud-based development tools (such as build integration, project planning etc.) must support integration with browser-based IDEs.

3. Browser-based IDEs should support interfaces allowing real-time capturing of a software developer’s activity in a software project to promote awareness among co-developers.
4. Browser-based IDEs must provide interfaces to enable integration with code forges in order to support open source software development in the cloud.

5. Integration between desktop-based and browser-based IDEs should be supported. This can include import and export of software projects as well as configuration settings.

With the advent of software development in the cloud, we see it a perfect example of capitalising on the approaches that are defined in this thesis and apply it to the cloud-based software development environments in order to enable a next generation of Linked Data Driven Software Development in Cloud (LD2SDiC).
BIBLIOGRAPHY


Reinhardt, W. (2009). Communication is the key - support durable knowledge sharing in software engineering by microblogging. In Software Engineering (Workshops), pages 329–340. (Cited on page 133.)


A.1 Socially Dependent Software Developer Pairs of Apache Projects

Figure A.1: Socially dependent software developer pairs of Apache Ant over the period of time.

Figure A.2: Socially dependent software developer pairs of Apache Camel over the period of time.
Figure A.3: Socially dependent software developer pairs of *Apache Directory* over the period of time.

Figure A.4: Socially dependent software developer pairs of *Apache Felix* over the period of time.
Figure A.5: Socially dependent software developer pairs of **APACHE HADOOP** over the period of time.

Figure A.6: Socially dependent software developer pairs of **APACHE LOGGING** over the period of time.
Figure A.7: Socially dependent software developer pairs of Apache Lucene over the period of time.

Figure A.8: Socially dependent software developer pairs of Apache Maven over the period of time.
Figure A.9: Socially dependent software developer pairs of **APACHE MINA** over the period of time.

Figure A.10: Socially dependent software developer pairs of **APACHE MYFACES** over the period of time.
A.2  SOCIALLY RELATED SOFTWARE DEVELOPER PAIRS OF APACHE PROJECTS

Figure A.11: Socially related software developer pairs of Apache Ant over the period of time.

Figure A.12: Socially related software developer pairs of Apache Camel over the period of time.
Figure A.13: Socially related software developer pairs of Apache Directory over the period of time.

Figure A.14: Socially related software developer pairs of Apache Felix over the period of time.
Figure A.15: Socially related software developer pairs of *Apache Hadoop* over the period of time.

Figure A.16: Socially related software developer pairs of *Apache Logging* over the period of time.
Figure A.17: Socially related software developer pairs of **APACHE LUCENE** over the period of time.

Figure A.18: Socially related software developer pairs of **APACHE MAVEN** over the period of time.
Figure A.19: Socially related software developer pairs of Apache Mina over the period of time.

Figure A.20: Socially related software developer pairs of Apache MyFaces over the period of time.
A.3 SOCIAL DEPENDENCIES AND SOCIAL INTERACTIONS AMONG SOFTWARE DEVELOPER PAIRS OF APACHE PROJECTS

![Socially Dependent Software Developer Pairs Over Time](image1)

![Socially Related Software Developer Pairs Over Time](image2)

Figure A.21: Social dependencies and social interactions among software developer pairs of Apache Ant over the period of time.
Figure A.22: Social dependencies and social interactions among software developer pairs of APACHE CAMEL over the period of time.
Figure A.23: Social dependencies and social interactions among software developer pairs of Apache Directory over the period of time.
Figure A.24: Social dependencies and social interactions among software developer pairs of *Apache Felix* over the period of time.
Figure A.25: Social dependencies and social interactions among software developer pairs of APACHE HADOOP over the period of time.
Figure A.26: Social dependencies and social interactions among software developer pairs of Apache Logging over the period of time.
Figure A.27: Social dependencies and social interactions among software developer pairs of Apache Lucene over the period of time.
Figure A.28: Social dependencies and social interactions among software developer pairs of Apache Maven over the period of time.
Figure A.29: Social dependencies and social interactions among software developer pairs of Apache Mina over the period of time.
Figure A.30: Social dependencies and social interactions among software developer pairs of Apache MyFaces over the period of time.
The links between mailing list and bug repository are identified by querying mailing list RDF datasets to retrieve resources having predicate ld2sd:mentioned with the value “Bug” as shown in Listing B.1.

```sparql
PREFIX ld2sd: <http://vocab.deri.ie/ld2sd#>
PREFIX sioc: <http://rdfs.org/sioc/types#>

SELECT count (distinct ?email)
WHERE {
    ?email a sioc:MailMessage .
    ?email ld2sd:mentioned "Bug" .
}
```

Listing B.1: A SPARQL query that counts the number of RDF links between mailing list and bug repository.

The links between mailing list and source control repository are identified by querying mailing list RDF datasets to retrieve resources having predicate ld2sd:mentioned with the value “SVNCommit” as shown in Listing B.2.

```sparql
PREFIX ld2sd: <http://vocab.deri.ie/ld2sd#>
PREFIX sioc: <http://rdfs.org/sioc/types#>

SELECT count (distinct ?email)
WHERE {
    ?email a sioc:MailMessage .
    ?email ld2sd:mentioned "SVNCommit" .
}
```

Listing B.2: A SPARQL query that counts the number of RDF links between mailing list and source control repository.

The links between mailing list and source code are identified by querying mailing list RDF datasets to retrieve resources having predicate ld2sd:mentioned with the value “JavaSource” as shown in Listing B.3.

```sparql
PREFIX ld2sd: <http://vocab.deri.ie/ld2sd#>
PREFIX sioc: <http://rdfs.org/sioc/types#>

SELECT count (distinct ?email)
WHERE {
    ?email a sioc:MailMessage .
    ?email ld2sd:mentioned "JavaSource" .
}
```

Listing B.3: A SPARQL query that counts the number of RDF links between mailing list and source code.
The links between source code and source control repository are identified by querying source control RDF datasets to retrieve resources having predicate owl:sameAs as shown in Listing B.4.

Listing B.4: A SPARQL query that counts the number of RDF Links between source control repository and source code.

```
1 PREFIX baetle: <http://baetle.googlecode.com/svn/ns/#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3
4 SELECT count ?javaclass
5 WHERE {
6   {?commit baetle:added ?source}
7   UNION
8   {?commit baetle:modified ?source} .
9   ?source owl:sameAs ?javaclass .
10 }
```

A list of hashtags that are used by software developers in tweets are identified by querying the Twitter RDF dataset as shown in Listing B.5.

Listing B.5: A SPARQL query that retrieves distinct hashtags used by software developers.

```
1 PREFIX moat: <http://moat-project.org/ns#>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3
4 SELECT distinct ?tag
5 WHERE {
8 }
```

A list of software developers who have used a particular hashtag are identified by querying the Twitter RDF dataset as shown in Listing B.6.

Listing B.6: A SPARQL query that retrieves all software developers who have used a particular hashtag.

```
1 PREFIX moat: <http://moat-project.org/ns#>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX dct: <http://purl.org/dc/terms/>
4
5 SELECT distinct ?dev
6 WHERE {
8   ?tweet moat:taggedWith "hadoop" .
10 }
```

A list of hashtags that are used by a software developer in his/her tweets are identified by querying the Twitter RDF dataset as shown in Listing B.7.

```
1 PREFIX moat: <http://moat-project.org/ns#>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX dct: <http://purl.org/dc/terms/>
4
5 SELECT distinct ?dev
6 WHERE {
10 }
```

A list of hashtags that are used by a software developer in his/her tweets are identified by querying the Twitter RDF dataset as shown in Listing B.7.
Listing B.7: A SPARQL query that returns a list of hashtags used by a software developer.

A list of tweets that mentions a particular hashtag along with its timestamp value are identified by querying the Twitter RDF dataset as shown in Listing B.8.

Listing B.8: A SPARQL query that returns a list of tweets that mention a particular hashtag.

Number of social interactions happened between two software developers on Twitter in a given period of time are identified by querying the Twitter RDF dataset as shown in Listing B.9.

Listing B.9: A SPARQL query that returns the number of times two software developers interacted with each other on Twitter in a certain time period.
Number of social interactions happened between two software developers on the Apache Maven mailing list in a given period of time are identified by querying the mailing list RDF datasets as shown in Listing B.10.

```
PREFIX email: <http://simile.mit.edu/2005/06/ontologies/email#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT distinct (?email) WHERE {
  ?email dct:date ?date .
  FILTER (xsd:dateTime(?date) >= "2003-05-03T18:34:54"^^xsd:dateTime
          && xsd:dateTime(?date) <= "2003-06-03T18:34:54"^^xsd:dateTime)
}
```

Listing B.10: A SPARQL query that returns the number of times two software developers interacted with each other on the mailing list of Apache Maven in a certain time period.

A list of source code files that have been committed to the source control repository in a given period of time are identified by querying the source control RDF datasets as shown in Listing B.11.

```
PREFIX baetle: <http://baetle.googlecode.com/svn/ns/#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT distinct ?file WHERE {
  ?commit rdf:type baetle:Committing .
  ?commit baetle:date ?date .
  FILTER (xsd:dateTime(?date) >= "2005-02-01T01:18:15"^^xsd:dateTime
          && xsd:dateTime(?date) <= "2005-03-01T01:18:15"^^xsd:dateTime)
}
```

Listing B.11: A SPARQL query that returns a list of source code files committed during a specific time period on source control repository.

A list of software developers that have committed a particular source code file (e.g., JasperException.java) to the source control repository in a specific time period are identified by querying the source control RDF datasets as shown in Listing B.12.
A list of emails that are sent in response to other emails are identified by querying the mailing list RDF datasets as shown in Listing B.13.

Listing B.13: A SPARQL query that return emails, which are response to other emails.

A list of email message-ids for which a particular email is a reply can be identified by querying the mailing list RDF datasets (considering Apache Tomcat as an example) as shown in Listing B.14.

Listing B.14: A SPARQL query that retrieve message-ids of those emails for which a particular email is a reply.

A list of software developers who have made commits to the source control repository are identified by querying the source control RDF datasets as shown in Listing B.15.
A list of source code files that have been committed to the source control repository by a particular software developer in a specific time period are identified by querying the source control RDF datasets (considering Apache Tomcat as an example) as shown in Listing B.16.

```
PREFIX baetle: <http://baetle.googlecode.com/svn/ns/#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT distinct ?file WHERE {
  ?commit rdf:type baetle:Committing .
  ?commit baetle:date ?date .
  FILTER (xsd:dateTime(?date) >= "2009-01-01T15:28:35"^^xsd:dateTime
  && xsd:dateTime(?date) <= "2009-02-01T15:28:35"^^xsd:dateTime)
}
```

Listing B.16: A SPARQL query that returns source code files committed to the source control repository by a software developer in a specific time period.

A list of distinct IDs that belongs to a software developer can be identified by querying all RDF datasets (considering Apache Tomcat as an example) as shown in Listing B.17.

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT distinct ?id WHERE {
  {<http://example.org/tomcat/author/kkolinko> owl:sameAs ?id .}
  ?id owl:sameAs <http://example.org/tomcat/author/kkolinko> .
}
```

Listing B.17: A SPARQL query that returns all IDs, which belongs to a software developer.
A partial list of SKOS concept scheme which defines the operating systems is shown in Listing C.1. Further, SKOS concepts are linked to the DBPEDIA dataset using owl:sameAs property in order to query further information about each concept.

```rml
@prefix base: <http://example.org/schemes/os#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

# Following is a partial list of operating system category extracted from SourceForge

base a skos:ConceptScheme;
  skos:prefLabel "Operating System"@en;
  dct:hasPart base: emulation;
  dct:hasPart base: modern;
  dct:hasPart base: embedded .

base: modern a skos:ConceptScheme ;
  skos:prefLabel "Modern (Vendor-Supported) Desktop Operating Systems"@en ;
  skos:hasTopConcept base: linux ;
  skos:hasTopConcept base: os-x ;
  skos:hasTopConcept base: solaris ;
  skos:hasTopConcept base: winxp .

base: linux a skos:Concept ;
  skos:inScheme base: ;
  skos:inScheme base: modern ;
  skos:prefLabel "Linux"@en ;
  owl:sameAs <http://dbpedia.org/resource/Linux> .

base: os-x a skos:Concept ;
  skos:inScheme base: ;
  skos:inScheme base: modern ;
  skos:prefLabel "OS X"@en ;
  owl:sameAs <http://dbpedia.org/resource/Mac_OS_X> .

base: solaris a skos:Concept ;
  skos:inScheme base: ;
  skos:inScheme base: modern ;
  skos:prefLabel "Solaris"@en ;
  owl:sameAs <http://dbpedia.org/resource/Solaris> .

base: winxp a skos:Concept ;
  skos:inScheme base: ;
  skos:inScheme base: modern ;
  skos:prefLabel "WinXP"@en ;
```

Listing C.1: SKOS Concept Scheme for operating system.
Listing C.2: RDF description of a GitHub project based on the data provided by FLOSSMOLE.

Listing C.3: RDF description of a GoogleCode project based on the data provided by FLOSSMOLE.
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfhloh: <http://rdfhloh.wikier.org/ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix rdfhloh: <http://rdfhloh.wikier.org/ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

217

<http://rdfohloh.wikier.org/project/filezilla> a doap:Project;
  rdfhloh:ohloh-page <http://www.ohloh.net/projects/filezilla>;
  doap:created "2006-11-26T00:20:40Z";
  doap:description ""FileZilla Client is a fast and reliable cross-platform FTP,
  FTPS and SFTP client with lots of useful features and an intuitive interface.
  FileZilla Server is a reliable FTP server."";
  doap:developer <http://rdfohloh.wikier.org/user/19797#person>,
    <http://rdfohloh.wikier.org/user/49123#person>;
  doap:download-page <http://filezilla-project.org/download.php>;
  doap:homepage <http://filezilla-project.org>;
  doap:name "FileZilla";
  doap:programming-language "C++";

<http://rdfohloh.wikier.org/project/filezilla.rdf> a foaf:Document;
  dct:isFormatOf <http://rdfohloh.wikier.org/project/filezilla>;
  rdfs:label "FileZilla's DOAP document serialized in RDF/XML";
<http://rdfohloh.wikier.org/user/19797#person> a foaf:Person;
  foaf:based_near [ geo:lat "51.165691";
    geo:long "10.451526" ];
  foaf:holdsAccount <http://rdfohloh.wikier.org/user/19797>;
  foaf:homepage <http://filezilla-project.org>;
  foaf:mbox_sha1sum "9925ebe2eb0f5d0a5de0b161ed7019ed020a48";
  foaf:name "Tim Kosse" .

Listing C.4: FileZilla project description by RDFOHLOH.
LIST OF FIGURES

Figure 1.1 Software project related information distributed on the Web in heterogeneous data sources. ................................................................. 3
Figure 1.2 An example of dereferencing a HTTP URI using content negotiation (picture taken from [Bizer et al., 2007]). .............................. 16
Figure 1.3 Snapshot of different data sources that contains information about APACHE LUCENE and are integrated to each other through owl:sameAs links. .... 18
Figure 1.4 A conceptual diagram depicting aggregated metrics value of a software project based on various metrics computed individually. .......... 20
Figure 2.1 An exemplary RDF statement representing “Aftab knows Michael”. ............................. 26
Figure 2.2 Pattern of interactions among software developers of squirrelmail project, taken from [Crowston and Howison, 2003]. ..................... 36
Figure 2.3 An exemplary call graph of a Java program. .............................................. 37
Figure 2.4 An exemplary socio dependencies graph showing information about software developers who are developing interdependent source code. .... 38
Figure 2.5 An exemplary social network graph showing dependencies among software developers. ................................................................. 38
Figure 3.1 Different software artifacts involved in the software development process, including its participants. We do not consider software artifacts that are highlighted in grey color while we present our integration approach later in this chapter. However, our approach can be extended and applied to all software artifacts involved in the software development process. 40
Figure 3.2 Graphical representation of a bug report in RDF by using classes and properties from baetle, foaf and dct RDF vocabularies. ................. 45
Figure 3.3 An example depicting connections among different software artifacts. Connections between software artifacts are made explicit by inserting RDF statements, which allows the user to navigate across various software artifacts. ................................................................. 46
Figure 3.4 A reference architecture of LD2SD methodology showing the 3 layers of data publishing & interlinking, indexing and exploitation of integrated information via end-user applications. .................................................... 49
Figure 3.5 A graphical DERI pipes editor that allows to fetch RDF data sources from user provided URLs, integrate and operate on them. ............... 51
Figure 3.6 Link discovery among software artifacts contained inside different software repositories hosting a software project. We show that similar software artifacts exist across different software repositories and forms an implicit connection between them, which is represented through dotted lines. ................................................................. 61
Figure 3.7  LD2SD interaction setup. The data from different software repositories are published as RDF and interlinked, yielding LD2SD datasets. Later, they are indexed by an indexing engine so that lookup service can be provided on top of interlinked datasets. Finally, deliver related information to the software developers in their IDEs.

Figure 3.8  Triggering the LD2SD plugin via the context menu.

Figure 3.9  Related information from different software repositories about a particular Java class displayed inside the Eclipse IDE.

Figure 3.10  Search results for a Java software package displayed in the Web interface.

Figure 3.11  Related information about a particular Java class that is queried automatically from different software repositories and displayed in the Web interface.

Figure 3.12  Average time spent by each participant on performing the tasks manually and through the LD2SD plugin.

Figure 4.1  Software developer communicating using multiple IDs on the mailing list. The white color nodes belong to a software developer (Yonik Seeley), who has used two different email IDs to communicate with others on the software project mailing list.

Figure 4.2  Comparison of different software developer identification approaches.

Figure 4.3  Bugs reported by software developers using multiple IDs.

Figure 4.4  Software developers existence in multiple Apache projects.

Figure 4.5  Software developer’s contribution in different Apache projects.

Figure 5.1  A conceptual diagram showing the transformation of database dumps into RDF datasets. The RDF datasets are later interlinked to each other based on metadata, similar software projects and software developers.

Figure 5.2  A conceptual diagram showing the transformation and integration of software developers and software projects related information from different open source repositories.

Figure 6.1  A software developer tweets about the Apache Maven 3.0.5 release that fixes a critical security bug. The tweet was captured from https://twitter.com/olamy/status/305334578103582720.

Figure 6.2  A conceptual diagram showing existence of software developers on social media channels and software repositories of a software project.

Figure 6.3  Distribution of the number of tweets sent per software developer.

Figure 6.4  Distribution of the number of times each hashtag is used by software developers.

Figure 6.5  Social network structure of software developers in different communication channels of an Apache software project. The nodes in both figures represent same software developers communicating with each other in different communication channels.

Figure 6.6  Social communication between 2 software developers on Twitter and mailing list in different periods of time.

Figure 6.7  Scatterplot of software developers communication with each other on Twitter and mailing list.

Figure 7.1  Socially dependent software developer pairs of Apache OrBiz over the period of time.
Figure 7.2  Socially related software developer pairs of Apache OfBiz over the period of time.

Figure 7.3  Social dependencies among three software developer pairs of Apache OfBiz over the period of time.

Figure 7.4  Social interactions among three software developer pairs of Apache OfBiz over the period of time.

Figure 7.5  Social dependency and social relation of a software developer with other software developers over the period of time.

Figure 7.6  Number of socially dependent software developers and those who actually had social interactions with him/her during the same time period.

Figure 8.1  Cloud-based software development stack.

Figure 8.2  Continuous integration and deployment of software applications in the cloud.

Figure A.1  Socially dependent software developer pairs of Apache ANT over the period of time.

Figure A.2  Socially dependent software developer pairs of Apache Camel over the period of time.

Figure A.3  Socially dependent software developer pairs of Apache Directory over the period of time.

Figure A.4  Socially dependent software developer pairs of Apache Felix over the period of time.

Figure A.5  Socially dependent software developer pairs of Apache Hadoop over the period of time.

Figure A.6  Socially dependent software developer pairs of Apache Logging over the period of time.

Figure A.7  Socially dependent software developer pairs of Apache Lucene over the period of time.

Figure A.8  Socially dependent software developer pairs of Apache Maven over the period of time.

Figure A.9  Socially dependent software developer pairs of Apache Mina over the period of time.

Figure A.10  Socially dependent software developer pairs of Apache MyFaces over the period of time.

Figure A.11  Socially related software developer pairs of Apache ANT over the period of time.

Figure A.12  Socially related software developer pairs of Apache Camel over the period of time.

Figure A.13  Socially related software developer pairs of Apache Directory over the period of time.

Figure A.14  Socially related software developer pairs of Apache Felix over the period of time.

Figure A.15  Socially related software developer pairs of Apache Hadoop over the period of time.

Figure A.16  Socially related software developer pairs of Apache Logging over the period of time.
Figure A.17  Socially related software developer pairs of APACHE LUCENE over the period of time. ................................................. 197
Figure A.18  Socially related software developer pairs of APACHE MAVEN over the period of time. ................................................. 197
Figure A.19  Socially related software developer pairs of APACHE MINA over the period of time. ................................................. 198
Figure A.20  Socially related software developer pairs of APACHE MyFaces over the period of time. ................................................. 198
Figure A.21  Social dependencies and social interactions among software developer pairs of APACHE ANT over the period of time. .......... 199
Figure A.22  Social dependencies and social interactions among software developer pairs of APACHE CAMEL over the period of time. ........ 200
Figure A.23  Social dependencies and social interactions among software developer pairs of APACHE DIRECTORY over the period of time. .... 201
Figure A.24  Social dependencies and social interactions among software developer pairs of APACHE FELIX over the period of time. .......... 202
Figure A.25  Social dependencies and social interactions among software developer pairs of APACHE HADOOP over the period of time. ........ 203
Figure A.26  Social dependencies and social interactions among software developer pairs of APACHE LOGGING over the period of time. .......... 204
Figure A.27  Social dependencies and social interactions among software developer pairs of APACHE LUCENE over the period of time. .......... 205
Figure A.28  Social dependencies and social interactions among software developer pairs of APACHE MAVEN over the period of time. .......... 206
Figure A.29  Social dependencies and social interactions among software developer pairs of APACHE MINA over the period of time. .......... 207
Figure A.30  Social dependencies and social interactions among software developer pairs of APACHE MyFaces over the period of time. ........ 208
### Table 1.1
Information related to Apache Lucene project that is available on the Web in heterogeneous data sources. ........................................... 7

### Table 1.2
Different types of software artifacts found in data sources. ............. 13

### Table 1.3
Reference implementation of URI patterns for different software artifacts found in each data source. ......................................................... 17

### Table 2.1
Description of SKOS vocabulary terms used in the thesis. ............... 29

### Table 2.2
Description of FOAF vocabulary terms used in the thesis. ............... 30

### Table 2.3
Description of SIOC vocabulary terms used in the thesis. ............... 31

### Table 2.4
Description of DCMI vocabulary terms used in the thesis. ............... 32

### Table 2.5
Description of MOAT vocabulary terms used in the thesis. ............... 32

### Table 2.6
Description of BAETLE vocabulary terms used in the thesis. ............... 33

### Table 2.7
Description of DOAP vocabulary terms used in the thesis. ............... 34

### Table 2.8
Different types of software artifacts found in each software repository. It shows that information about software artifacts is distributed across different software repositories and there exists an implicit connection between the information contained inside these software repositories. .... 41

### Table 2.9
RDF vocabularies and their exemplary properties that are used to represent software artifacts as RDF statements. ........................................... 44

### Table 2.10
URI patterns for different types of entities found in Java source code. ... 53

### Table 2.11
URI patterns for different types of entities found in a source control commit log. ................................................................. 55

### Table 2.12
URI patterns for different types of entities found in a bug report. ......... 57

### Table 2.13
URI patterns for different types of entities found in an email. ............. 59

### Table 2.14
Apache projects data range. .......................................................... 64

### Table 2.15
An overview of raw data sources extracted from Apache projects. .... 65

### Table 2.16
Total number of RDF triples generated for each software repository of different Apache projects. .......................................................... 65

### Table 3.1
Different types of software artifacts found in each software repository. It shows that information about software artifacts is distributed across different software repositories and there exists an implicit connection between the information contained inside these software repositories. .... 41

### Table 3.2
RDF vocabularies and their exemplary properties that are used to represent software artifacts as RDF statements. ........................................... 44

### Table 3.3
URI patterns for different types of entities found in Java source code. ... 53

### Table 3.4
URI patterns for different types of entities found in a source control commit log. ................................................................. 55

### Table 3.5
URI patterns for different types of entities found in a bug report. ......... 57

### Table 3.6
URI patterns for different types of entities found in an email. ............. 59

### Table 3.7
Apache projects data range. .......................................................... 64

### Table 3.8
An overview of raw data sources extracted from Apache projects. .... 65

### Table 3.9
Total number of RDF triples generated for each software repository of different Apache projects. .......................................................... 65

### Table 3.10
Cross linking between RDF datasets of different software repositories of Apache projects. ................................................................. 66

### Table 3.11
Size of dataset used for evaluation. .................................................. 74

### Table 3.12
Familiarity of the participants with the software tools/repositories that were used in the evaluation. .................................................. 74

### Table 3.13
Participants feedback after using the LD2SD plugin for Eclipse IDE. ... 75

### Table 3.14
Relevant information about some selected Apache Tomcat software packages which exists in different software repositories and is automatically retrieved using LD2SD plugin. .................................................. 78

### Table 4.1
Different types of IDs found in software repositories [Robles and Gonzalez-Barahona, 2005]. It also shows that the software developer IDs in different software repositories are represented in different formats, which makes it difficult to interlink them to each other. .................................................. 82

### Table 4.2
Names-email address similarity results. ........................................... 85
Table 4.3 Email address similarity results. ................................................................. 86
Table 4.4 Namescheme - email address results. ...................................................... 87
Table 4.5 Apache projects data range. ..................................................................... 94
Table 4.6 Matched IDs of software developers based on mailing list repository. .... 94
Table 4.7 Matched IDs of software developers based on bug repository. .............. 95
Table 4.8 Matched IDs of software developers across Apache projects based on mailing list repository. ................................................................. 96
Table 4.9 Matched IDs of software developers across Apache projects based on bug repository. ................................................................. 96
Table 5.1 Total number of software projects and software developers found in the FLOSSMOLE database dumps of different code forges. ......................... 110
Table 5.2 Interlinking code forge metadata terms with DBpedia dataset. ............... 111
Table 5.3 Number of software projects developed using different programming languages on code forges. ................................................................. 111
Table 5.4 Geo location and rank of a software developer retrieved from OHIOH. .... 113
Table 5.5 Overlapping between code forges based on the existence of software developers on multiple code forges. ................................................................. 113
Table 5.6 Matching similar software projects across different code forges based on their URLs. ................................................................. 115
Table 5.7 List of software developers working on the same software project across different code forges. ................................................................. 116
Table 5.8 Matching software developers based on similar software projects across different code forges. ................................................................. 117
Table 5.9 List of software projects on which a software developer is working across different code forges. ................................................................. 118
Table 5.10 URI patterns for different types of entities found in GitHub. ................. 121
Table 5.11 URI patterns for different types of entities found in OHIOH. ............... 124
Table 5.12 A list of software projects from different code forges on which a particular software developer of Apache ANT is working or has worked in the past. 127
Table 5.13 List of source control commits made to a particular software project by a software developer using different programming languages. ............... 128
Table 5.14 Average commit ratio of a software developer on different programming languages. ................................................................. 129
Table 6.1 URI pattern for an entity found in a Twitter user profile. ....................... 135
Table 6.2 URI patterns for different types of entities found in a tweet. ................. 136
Table 6.3 Software developers contributing to Apache software projects and also using Twitter. ................................................................. 138
Table 6.4 Twitter dataset summary. ................................................................. 138
Table 6.5 Most commonly used hashtags by software developers. ....................... 141
Table 6.6 Most commonly used hashtags by software developers. We also show if software developers have ever used software project related hashtags while tweeting. ................................................................. 141
Table 6.7 Average number of days taken by software developers to post a software project related tweet. We also show number of tweets in which software project related hashtag is used by software developers. ............... 143
Table 6.8 Mean and Median of correlation values. . . . . . . . . . . . . . . . . . . . 149
<table>
<thead>
<tr>
<th>Listing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Turtle representation of RDF statements. We have also used 2 popular vocabularies in this example rather than only using “example” namespace.</td>
</tr>
<tr>
<td>2.2</td>
<td>A SPARQL query that returns all people who were born in Ireland, ordered by name.</td>
</tr>
<tr>
<td>3.1</td>
<td>Java source code snippet taken from the Apache Tomcat project.</td>
</tr>
<tr>
<td>3.2</td>
<td>RDF representation of Java source code based on Listing 3.1.</td>
</tr>
<tr>
<td>3.3</td>
<td>Source control commit log snippet.</td>
</tr>
<tr>
<td>3.4</td>
<td>RDF representation of a source control commit log based on Listing 3.3.</td>
</tr>
<tr>
<td>3.5</td>
<td>Bug report snippet taken from the Apache Tomcat project.</td>
</tr>
<tr>
<td>3.6</td>
<td>RDF representation of a bug report based on Listing 3.5.</td>
</tr>
<tr>
<td>3.7</td>
<td>Email snippet taken from the Apache Tomcat mailing list archive.</td>
</tr>
<tr>
<td>3.8</td>
<td>RDF representation of an email based on Listing 3.7.</td>
</tr>
<tr>
<td>3.9</td>
<td>An interlinking example connecting source control commit log and a bug report.</td>
</tr>
<tr>
<td>3.10</td>
<td>An interlinking example connecting source control commit log and Java source code.</td>
</tr>
<tr>
<td>3.11</td>
<td>An interlinking example connecting an email and source control commit log.</td>
</tr>
<tr>
<td>3.12</td>
<td>An interlinking example connecting an email and a bug report.</td>
</tr>
<tr>
<td>3.13</td>
<td>SPARQL query that counts the number of RDF links between source control and bug repository RDF datasets.</td>
</tr>
<tr>
<td>3.14</td>
<td>Exemplary SPARQL query that retrieve bugs, which have been fixed by &quot;JasperException.java&quot;.</td>
</tr>
<tr>
<td>4.1</td>
<td>RDF representation of an email.</td>
</tr>
<tr>
<td>4.2</td>
<td>RDF representation of a bug report.</td>
</tr>
<tr>
<td>4.3</td>
<td>An interlinking example connecting different IDs of a software developer.</td>
</tr>
<tr>
<td>5.1</td>
<td>An exemplary SKOS concept scheme that defines different operating systems for SourceForge.</td>
</tr>
<tr>
<td>5.2</td>
<td>Adding a new concept to the existing SKOS concept scheme.</td>
</tr>
<tr>
<td>5.3</td>
<td>An interlinking example connecting a metadata term to its relevant entity in DBpedia.</td>
</tr>
<tr>
<td>5.4</td>
<td>RDF representation of a SourceForge software project based on the data provided by FLOSSMOLE.</td>
</tr>
<tr>
<td>5.5</td>
<td>A SPARQL query that retrieves information about a software developer working on FileZilla project from Ohloh.</td>
</tr>
<tr>
<td>5.6</td>
<td>An interlinking example connecting similar software project on GoogleCode and SourceForge.</td>
</tr>
<tr>
<td>5.7</td>
<td>A SPARQL query that retrieves a list of software developers working on the same software project (i.e., quadra) across different code forges.</td>
</tr>
<tr>
<td>5.8</td>
<td>An interlinking example connecting IDs of the same software developer found in GoogleCode and SourceForge.</td>
</tr>
<tr>
<td>Listing</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>5.9</td>
<td>A SPARQL query that retrieves all software projects of a software developer across different code forges.</td>
</tr>
<tr>
<td>5.10</td>
<td>RDF representation of a software project information extracted from GitHub.</td>
</tr>
<tr>
<td>5.11</td>
<td>RDF representation of a software developer profile extracted from GitHub.</td>
</tr>
<tr>
<td>5.12</td>
<td>RDF representation of a software project information extracted from GitHub.</td>
</tr>
<tr>
<td>5.13</td>
<td>RDF representation of a software developer profile extracted from GitHub.</td>
</tr>
<tr>
<td>5.14</td>
<td>An interlinking example connecting software developer and software project IDs found in different open source repositories.</td>
</tr>
<tr>
<td>5.15</td>
<td>A query that lists all software projects on which a software developer of Apache ANT is working in different code forges.</td>
</tr>
<tr>
<td>5.16</td>
<td>A SPARQL query that retrieves number of commits made to a particular software project by a software developer using different programming languages.</td>
</tr>
<tr>
<td>5.17</td>
<td>A SPARQL query that retrieves average commit ratio of a software developer for different programming languages based on his development activity in different software projects.</td>
</tr>
<tr>
<td>6.1</td>
<td>RDF representation of a software developer’s Twitter profile.</td>
</tr>
<tr>
<td>6.2</td>
<td>RDF representation of a tweet.</td>
</tr>
<tr>
<td>6.3</td>
<td>RDF representation of an email which is a reply to another email.</td>
</tr>
<tr>
<td>B.1</td>
<td>A SPARQL query that counts the number of RDF links between mailing list and bug repository.</td>
</tr>
<tr>
<td>B.2</td>
<td>A SPARQL query that counts the number of RDF links between mailing list and source control repository.</td>
</tr>
<tr>
<td>B.3</td>
<td>A SPARQL query that counts the number of RDF links between mailing list and source code.</td>
</tr>
<tr>
<td>B.4</td>
<td>A SPARQL query that counts the number of RDF Links between source control repository and source code.</td>
</tr>
<tr>
<td>B.5</td>
<td>A SPARQL query that retrieves distinct hashtags used by software developers.</td>
</tr>
<tr>
<td>B.6</td>
<td>A SPARQL query that retrieves all software developers who have used a particular hashtag.</td>
</tr>
<tr>
<td>B.7</td>
<td>A SPARQL query that returns a list of hashtags used by a software developer.</td>
</tr>
<tr>
<td>B.8</td>
<td>A SPARQL query that returns a list of tweets that mention a particular hashtag.</td>
</tr>
<tr>
<td>B.9</td>
<td>A SPARQL query that returns the number of times two software developers interacted with each other on Twitter in a certain time period.</td>
</tr>
<tr>
<td>B.10</td>
<td>A SPARQL query that returns the number of times two software developers interacted with each other on the mailing list of Apache Maven in a certain time period.</td>
</tr>
<tr>
<td>B.11</td>
<td>A SPARQL query that returns a list of source code files committed during a specific time period on source control repository.</td>
</tr>
<tr>
<td>B.12</td>
<td>A SPARQL query that returns software developers who have committed a particular source code in a specific time period.</td>
</tr>
<tr>
<td>B.13</td>
<td>A SPARQL query that return emails, which are response to other emails.</td>
</tr>
<tr>
<td>B.14</td>
<td>A SPARQL query that retrieve message-ids of those emails for which a particular email is a reply.</td>
</tr>
<tr>
<td>B.15</td>
<td>A SPARQL query that returns software developers who have made commits to the source control repository.</td>
</tr>
</tbody>
</table>
Listing B.16  A SPARQL query that returns source code files committed to the source control repository by a software developer in a specific time period. ........................................ 214
Listing B.17  A SPARQL query that returns all IDs, which belongs to a software developer. 214
Listing C.1   SKOS Concept Scheme for operating system. ........................................ 215
Listing C.2   RDF description of a GitHub project based on the data provided by FLOSSMOLE. 216
Listing C.3   RDF description of a GOOGLECODE project based on the data provided by FLOSSMOLE. ........................................ 216
Listing C.4   FileZilla project description by RDFOHLOR. .................................... 217