Understanding the Evolution of Socio-technical Aspects in Open Source Ecosystems: An Empirical Analysis of GNOME

Mathieu Goeminne

A dissertation submitted in fulfillment of the requirements of the degree of Docteur en Sciences

Advisor
Dr. Tom Mens
Université de Mons, Belgium

Jury
Dr. Xavier Blanc
Université de Bordeaux 1, France

Dr. Véronique Bruyère
Université de Mons, Belgium

Dr. Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos, Spain

Dr. Tom Mens
Université de Mons, Belgium

Dr. Alexander Serebrenik
Technische Universiteit Eindhoven, The Netherlands

Dr. Jef Wijsen
Université de Mons, Belgium

June 2013
Acknowledgments

This work could have never been completed without the support of many people. First things first, I would like to thank Dr. Tom Mens for his help throughout the years. He provided me advice and support, and helped me to grow up professionally and personally. He forced me to surpass myself, and I would like to thank him warmly for that. I would thank the other members of my jury: Dr. Xavier Blanc, Dr. Véronique Bruyère, Dr. Jesus M. Gonzalez-Barahona, Dr. Alexander Serebrenik, and Dr. Jef Wijsen. All of them have positively influenced my vision of the world and left their mark on me. Thank you for the time you have given me. I hope I will have the pleasure to continue to work with you in the future.

I would also like to thank Bogdan Vasilescu, who co-authored a journal article from which a chapter of this dissertation has been extracted. Thank you for your collaboration and your precious advices.

I wish to thank my friends and my colleagues who offered me so many joy and who regularly proved me that there is a life apart from the thesis! Special thanks go to my family for their encouragement and for being present in my life. To all of you: Thank you for being on my side during this crazy adventure.

Finally, I would like to thank my wife Nadège and my son Antoine. I’m sorry for the too many hours I have passed far away, even when I was sitting close to you. Thank you for your unconditional support.
Abstract

Since the 70’s, software development has experienced an exponential growth. The number of developed software products, their size and their complexity has become so important that understanding their functioning and managing their evolution have become very hard today. Open source software (OSS) does not escape from this growth and the problems it raises. For more than a decade, OSS systems have been the subject to an increasing interest from the academic community, individuals and the software industry at large, and their development is booming because of their low cost of use (OSS systems are generally freely available), their low barriers to entry for the developers, their low cost of development (they may be built by reusing other OSS systems), and the large quantity of easily available historical data.

Contrary to the traditional commercial and proprietary software, OSS is typically developed by a group of persons dispersed all over the world. This geographical distribution forces contributors to use tools allowing an asynchronous communication and an information exchange over big space scales. The public availability of the historical data being handled by these tools facilitates the analysis of OSS evolution.

Initially, empirical analysis of OSS projects evolution was limited to the study of source code evolution only. Later, other software development artefacts have been taken into account as well. For instance, the first analyses of OSS project mailing lists date to 2002 [157]. However, the main factor that drives the evolution of a software project is the people contributing to it. Hence, in order to better comprehend how OSS projects evolve, one needs to gain a better insight in the socio-technical aspects that surrounding them. In order to get a more accurate model of the interaction between the project contributors one needs to consider development artefacts that contain information about its social aspects, such as bug reports, e-mail discussions and version commits.

Frequently, collections of different projects are developed and evolve together in the same environment. We refer to these collections as software ecosystems. Since the contributors to the projects belonging to these ecosystems work together towards a common
goal, they tend to form de facto communities. It is therefore important to study the social aspects not only at the level of individual projects, but also at the level of the entire ecosystem.

The goal of this dissertation is to understand the evolution of the social aspects in open source ecosystems. More precisely, we study how contributors to open source ecosystems can be grouped in different communities that evolve and collaborate in different ways. In doing so, we provide evidence that contributors have specificities that are not taken into account by today’s analysis tools. Becoming aware of these specificities opens up new research and practically relevant questions on how new automated tools can be designed and used to offer better support to the ecosystem’s contributors in their activities.

The contributions of this dissertation are manifold. We developed an application framework that allows us to empirically study the evolution of software ecosystems. Focusing on the GNOME ecosystem, we designed a systematic approach for detecting the multiple accounts used by contributors to access the software repositories and used it to gain a better insight in the communities belonging to the ecosystem. We defined objective criteria according to which these contributors can be categorised. In the GNOME history we observed a power law behaviour between the number of contributors and their contributions, in term of commits submitted, mails sent and bug reports handled. With further statistical analyses we established correlations and trends between the contributors’ effort, their favourite means of communication and the activity types in which they are involved. For example, we observed that the contributors tend to restrict themselves to a limited number of activity types, but the more active a contributor is, the more he tends to spread his effort over different types of activity. When studying the evolution of GNOME contributors, we observed a tendency of specialisation towards less activity types. We also observed that, during the last years, the effort in each of the studied activity types is decreasing.
Résumé

Depuis les années 70, le développement logiciel connaît une croissance exponentielle. Le nombre de produits logiciels développés, leur taille et leur complexité sont devenus si importants que la compréhension de leur fonctionnement et la gestion de leur évolution sont devenues très difficiles de nos jours. Les logiciels open source (OSS) n’échappent pas à cette croissance ni aux problèmes qu’elle pose. Depuis plus d’une décennie, les systèmes open source font l’objet d’un intérêt croissant de la communauté académique, des particuliers et de l’industrie logicielle en général. Leur développement explode du fait de leur faible coût d’utilisation (les systèmes open source sont généralement librement accessibles), leur faible ticket d’entrée pour les développeurs, leur faible coût de développement (ils peuvent être construits en réutilisant d’autres systèmes open source), ainsi que la grande quantité de données historiques pouvant être aisément obtenues.

Contrairement aux logiciels commerciaux et propriétaires traditionnels, les logiciels open source sont typiquement développés par un groupe de personnes dispersées à travers le monde. Cette distribution géographique oblige les contributeurs à utiliser des outils permettant une communication asynchrone et l’échange d’informations sur de grandes distances. La mise à disposition publique des données historiques gérées par ces outils facilite l’analyse de l’évolution des logiciels open source.

Initialement, l’analyse empirique de l’évolution des projets open source était limitée à l’étude de l’évolution du code source. Par la suite, d’autres artefacts de développement logiciel ont été pris en compte. Par exemple, les premières analyses des listes de diffusion des projets open source datent de 2002 [157]. Cependant, les personnes contribuant à un projet logiciel en constituent le principal vecteur d’évolution. Ainsi, afin de mieux comprendre la manière dont les projets open source évoluent, il est nécessaire d’avoir un meilleur aperçu des aspect socio-techniques qui les entourent. Afin d’avoir un modèle plus précis et plus juste des interactions existant entre les contributeurs du projet, il est nécessaire de considérer les artefacts de développement qui contiennent de l’information
relative à ses aspects sociaux, tels que les rapports d’erreur, les discussions par e-mail et les commits de version.

Fréquemment, des projets logiciels sont développés et évoluent ensemble dans le même environnement. Nous appelons de telles collections de projets des écosystèmes logiciels. Dans la mesure où les contributeurs des projets appartenant à ces écosystèmes travaillent ensemble dans un but commun, ils ont tendance à former de facto des communautés. Il est donc important d’étudier les aspects sociaux non seulement au niveau des projets individuels, mais également au niveau de l’écosystème dans son ensemble.

L’objectif de cette thèse est de comprendre l’évolution des aspects sociaux des écosystèmes open source. Plus précisément, nous étudions la manière dont les contributeurs impliqués dans les écosystèmes open source peuvent être groupés en différentes communautés qui évoluent et collaborent de différentes manières. De la sorte, nous apportons des indices probants selon lesquels les contributeurs ont des spécificités qui ne sont pas prises en compte par les outils d’analyses actuels. La prise de conscience de ces spécificités laisse entrevoir de nouvelles questions de recherche et de nouvelles pratiques sur la manière de concevoir de nouveaux outils automatisés aidant plus efficacement les contributeurs de l’écosystème dans la réalisation de leurs activités.

Les contributions de cette thèse sont multiples. Nous avons développé un framework applicatif qui permet la réalisation d’études empiriques des écosystèmes logiciels. Concentrant nos efforts sur l’écosystème GNOME, nous avons conçu une approche systématique pour la détection des multiples comptes utilisés par les contributeurs pour accéder aux dépôts logiciels. Nous avons utilisé cette approche pour pouvoir établir un meilleur modèle des communautés impliquées dans l’écosystème. Dans l’historique de GNOME, nous avons observé des lois de puissance entre le nombre de contributeurs et leurs contributions, en terme de commits soumis, d’e-mails envoyés et de rapports d’erreur gérés. Des analyses statistiques plus détaillées nous ont permis d’établir la présence de corrélations et de tendances entre l’effort réalisé par les contributeurs, leurs moyens de communication préférés et les types d’activité dans lesquels ils sont appliqués. Par exemple, nous avons observé que les contributeurs tendent à se restreindre à un nombre limité de types d’activité, mais aussi que plus un contributeur est actif, plus il a tendance à répartir son effort sur différents types d’activité. Lors de l’étude de l’évolution des contributeurs de GNOME, nous avons constaté que ceux-ci ont tendance à se spécialiser en un nombre réduit de types d’activité. Nous avons également observé qu’au cours de ces dernières années, l’effort consenti dans chacun des types d’activité étudiés décroît avec le temps.
# Table of Contents

I The thesis 1

1 Introduction 3

1.1 Research Goals and Contributions 4

1.2 Context 6

1.2.1 Open Source Software 6

1.2.2 Software Ecosystems 8

1.2.3 Social Aspects of Software Ecosystems 8

1.2.4 GNOME 9

1.3 Structure of this thesis 13

2 State of the Art 17

2.1 Evolution of Open Source Software 18

2.2 Social Aspects of Software 19

2.3 GNOME 22
## II Tooling

### 3 An application framework for studying OSS ecosystems

#### 3.1 Introduction

#### 3.2 The framework

- **3.2.1 Overall architecture**
- **3.2.2 Extraction layer**
- **3.2.3 Data post-processing layer**
- **3.2.4 Application Layer**

#### 3.3 Conclusion

### 4 Identity merging

#### 4.1 Introduction

#### 4.2 Terminology

#### 4.3 Methodology

- **4.3.1 Software project selection**
- **4.3.2 Data extraction**
- **4.3.3 Reference merge model**
- **4.3.4 Algorithm comparison**

#### 4.4 Identity merge algorithms

- **4.4.1 Simple algorithm**
- **4.4.2 Bird’s algorithm**
### III Empirical studies

6 Evidence for the Pareto principle in OSS activity

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Introduction</td>
<td>80</td>
</tr>
<tr>
<td>6.2 Related Work</td>
<td>80</td>
</tr>
<tr>
<td>6.3 Methodology</td>
<td>82</td>
</tr>
<tr>
<td>6.3.1 GQM paradigm</td>
<td>82</td>
</tr>
<tr>
<td>6.3.2 Basic metrics</td>
<td>82</td>
</tr>
<tr>
<td>6.3.3 Econometrics</td>
<td>83</td>
</tr>
<tr>
<td>6.4 Experimental setup</td>
<td>86</td>
</tr>
<tr>
<td>6.4.1 Implementation</td>
<td>86</td>
</tr>
<tr>
<td>6.4.2 On the Pareto principle and power law behaviour</td>
<td>87</td>
</tr>
<tr>
<td>6.4.3 Selected projects</td>
<td>87</td>
</tr>
<tr>
<td>6.5 Empirical study</td>
<td>88</td>
</tr>
<tr>
<td>6.5.1 Activity distribution</td>
<td>88</td>
</tr>
<tr>
<td>6.5.2 Activity overlap</td>
<td>92</td>
</tr>
<tr>
<td>6.5.3 Evolution over time</td>
<td>94</td>
</tr>
<tr>
<td>6.6 Discussion</td>
<td>96</td>
</tr>
<tr>
<td>6.7 Threats to validity</td>
<td>99</td>
</tr>
<tr>
<td>6.8 Conclusion</td>
<td>100</td>
</tr>
</tbody>
</table>
7 On the specialisation of GNOME workload 103

7.1 Introduction ......................................................... 104

7.2 Methodology .......................................................... 105
  7.2.1 Research Goals ............................................. 105
  7.2.2 Identity merging ............................................. 106
  7.2.3 Activity types detection ................................. 110
  7.2.4 Metrics ...................................................... 110
  7.2.5 Data analysis ............................................... 114
  7.2.6 procedure and $\overline{T}$-graph ...................... 116

7.3 Goal 1: How does workload vary across projects? .......... 119
  7.3.1 How does project workload vary across the ecosystem? 120
  7.3.2 Which projects are more active? ......................... 121
  7.3.3 How specialised are projects towards different activity types? 122
  7.3.4 What are the characteristics of specialised projects? 127
  7.3.5 Summary for Goal 1 ........................................ 133

7.4 Goal 2: How does workload vary across authors? .......... 133
  7.4.1 How does workload vary across authors? ............... 134
  7.4.2 Which kind of authors are more active? ............... 135
  7.4.3 How specialised are authors towards different activity types? 136
  7.4.4 What are the characteristics of specialised authors? 144
  7.4.5 Summary for Goal 2 ........................................ 149

7.5 Threats to validity ................................................ 149

7.6 Related work ....................................................... 151
  7.6.1 Studies of open source software contributors .......... 151
  7.6.2 Studies of the GNOME ecosystem ...................... 152

7.7 Conclusion .......................................................... 153
Part I

The thesis
Similarly to biological ecosystems, software ecosystems are made of entities (in that case, software systems) that co-evolve to create a unified environment [112]. Because these systems are developed jointly, humans responsible of their evolutions have to collaborate (and therefore, to communicate). This dissertation contributes to the research in this domain by empirically studying the evolution of the socio-technical aspects of a popular open source ecosystem.

After a description of the concepts involved in this research domain, this chapter exposes the main contributions of our dissertation and presents the global organisation.

This introductory chapter is partially inspired by a chapter from *Software Ecosystems* [63] and an article published in the Empirical Software Engineering Journal [172].
1.1 Research Goals and Contributions of this Thesis

Social aspects inherently take an important part in the evolution of software ecosystems. This is particularly true for open source ecosystems where, even if some contributions are provided by developers paid by companies, most of the contributions are made by volunteers who can join and leave the ecosystem’s projects at will. In this volatile environment presenting an important turnover of contributors, communication and social organisation are the key elements for a good knowledge transmission and for developer retention. Consequently, the long term goal of our research consists of supporting communities involved in such ecosystems by providing tools for better understand the evolution of the socio-technical aspects of these ecosystems.

To achieve this goal we need to develop tools offering a way for the communities to improve their efforts. For instance, these tools can take the form of dashboards providing synthetic and/or graphical views to developers and project leaders about general trends and potential issues in the system they develop. Using these dashboards, community members can more easily prevent technical or social problems before the software quality turns sour. Some dashboards have already been proposed in research literature, such as SECONDA [131], The Small Project Observatory [106] and Complicity [124]. However, these tools mainly focus on visualisations and only provide basic metrics related to software evolution and its social aspects. These limitations can be overcome by providing advanced metrics and statistical tools.

Before we can even start to build tools that provide a relevant information and a helpful support to the communities involved in a software ecosystem, a preliminary understanding of how these communities work and interact together over time is required. This understanding can be reached by extensive empirical analyses of evolution of ecosystems based on the collection, extraction and post-processing of historical data of studied ecosystems.

This dissertation focuses on the last objectives leading to the long term research goal, that is to say:

1. Data extraction and post-processing from open source ecosystems. Post-processing is needed to deal with inconsistent and incomplete data. It also enables us to make explicit links between multiple data sources that do not respect the same conventions. Results of post-processing are recorded for further analysis.

2. Empirical analysis and interpretation of the extracted data;
3. Understanding of social interactions in open source ecosystems.

In order to meet our three objectives, the dissertation addresses several research issues related to the social context in which software ecosystems evolve over time. Each of them will be detailed and refined later in this dissertation.

- As a case study we studied an active, long-lived, popular open source software ecosystem consisting of a large number of software projects together with its communities of developers, mailers and bug maintainers.

- In order to study social aspects of open source ecosystems, we needed to combine and integrate data originating from different data sources. To achieve this we developed a generic application framework and we instantiated this framework to carry out empirical research on the selected ecosystem. We implemented the framework by combining existing and self-made tools, enabling us to extract and analyse data from the selected software ecosystem. The framework was implemented in such a way that it can be easily adapted to other open source ecosystems.

- We tried to overcome the limitations of today's tools available for the analysis of open source ecosystem. These limitations are due to the fact that current tools are generally made to support a single project, or even a single aspect of a single project while we want to deal with complete ecosystems. The history of a software evolution is therefore dispersed over several tools. The same physical person often uses different accounts in different tools to develop and evolve the ecosystem (for instance, source code repositories, mailing lists systems and bug trackers). The social aspects of open source software being our main concern, we focused on identifying accounts that represent the same physical person. This activity was refined in the following sub-activities:
  
  - Develop an identity merge tool to detect and merge all the accounts belonging to the same physical person in a given data source. Depending on the type of data source, the detection is based on several heuristics using available information such as user name, e-mail address, etc. Because these heuristics do not provide an exact answer to the detection problem a priori, their efficiencies must be compared.

  - Extend the identity merge tool to deal with accounts belonging to the same physical person in multiple data sources. This sub-activity extends the previous one by taking into account more than one data source at a time. The heuristics used in the previous sub-activity must be adapted to take into account the fact that considered data sources can be of different types and handle different kinds of information.
The identity merging was applied to the source code repositories of all available projects in the selected ecosystem and to the mail discussions and bug reports of a subset of these projects.

- We analysed the effectiveness of the developed identity merge tool by discussing and qualitatively comparing existing and original solutions. The case study used for the comparison was based on a selection of the available projects.

- Study the contributors’ involvement among several activity types such as coding, translating and documenting. If coding is the most studied (and probably the most popular) activity, there exist numerous other tasks to improve the ecosystem. In order to understand the actual contributions of persons involved in a software ecosystem, we need to classify these contributors into sub-communities according to the type of changes made on the projects in which they are involved.

The above research issues are addressed in this dissertation by studying the type of data source in which contributors are involved. We considered source code repositories, mailing lists and bug trackers of some projects and studied the evolution of their involvements over time. We also designed an automated method for categorising each file added to a source code repository according to the activity type associated to the file and studied the contributors specialisation and involvement in the projects of the selected ecosystem.

1.2 Context

1.2.1 Open Source Software

Software systems are among the most complex artefacts ever created by humans. Collaborative development of free software has witnessed an exponential increase in the last two decades. It represents a successful model of software development where communities of developers collaborate on an often voluntary basis, while users and developers of the software systems can generally submit bug reports and requests for changes, and often need to be kept satisfied in order to maintain their involvement in the system.

In the scientific literature we can find two different but similar notions of openness or freeness: free software and open source software. The first type of software emphasises a precise goal: the freedom to use, analyse, modify, and distribute the software. Because of the ambiguity of the word *free* in English language, the term *Libre software* is also used to refer to the same concept. This type of software takes close interest in the practical
implications of this freedom: the source code availability for reusing software components and sharing the development effort between the different stakeholders involved in its creation and its maintenance. Each of these points of view is supported by a non-profit association. For the free software, this is the Free Software Foundation (FSF), whereas the open source software is advocated by the Open Source Initiative (OSI).

The FSF promotes the freedom to use software and to defend the rights of the free software users. According to the FSF, the term *free software* refers to four degrees of freedom: (1) freedom to execute a program, for any (private or commercial) usage; (2) freedom to access the source code in order to study the software functioning and to adapt it to one’s own needs; (3) freedom to redistribute copies of the program; (4) freedom to distribute the modifications made to the program. This last freedom, which necessitates the accessibility to the source code, gives to the whole community the opportunity to get access to all changes and improvements made to the program.

The OSI has been created to inform and promote the benefits of open code. It also facilitates the setting-up of links among the components of the open source community. According to the OSI, open code constitutes a software development model that combines the power of revision among developers with a transparency of the process of software development. In the opinion of this association, the promise of open source is better quality and higher reliability, more flexibility, lower cost, and lesser dependence on commercial software providers. One of the main activities of the OSI is to maintain the definition of open source for the good of the community (in a broad sense).

The main difference between the points of view of the FSF and the OSI is the targeted goal. Since the FSF puts the emphasis on the *why* (that is, the promotion and the defense of the freedom), the OSI mostly focuses on the *how* (that is, the availability of the source code and the followed development process).

In the context of this thesis, the differences between free and open source software are considered of minor importance and the term *Open Source Software* (OSS) is used to describe a software system developed accordingly to these notions. We use both of the terms *closed source* and *proprietary* software to refer to non-open source software. Even if these terms cover distinct concepts, their specificities are irrelevant for the purpose of our dissertation.

The focus of this dissertation is on OSS systems for the following reasons: (i) the increasing popularity of OSS, even in industry; (ii) the abundance and accessibility of software projects for which historical data is freely available; (iii) the ability

---

1. www.fsf.org
2. www.opensource.org
to publish scientific results about these systems without breaching confidentiality agreements; (iv) the ability to allow other researchers to verify and reproduce the obtained results.

1.2.2 Software Ecosystems

A biological ecosystem is defined as a biological environment consisting of all species living together and co-evolving in a particular area, as well as the physical components of the environment or habitat with which the organisms interact (e.g., air, soil, water and sunlight) [180]. Drawing the analogy with biological ecosystems, collections of software projects developed by the same developer community and interacting together can be considered as software ecosystems. While being significantly different, we can observe many similarities between both types of ecosystems: a software ecosystem is made up of a coherent collection of software projects (including software and hardware resources) that are developed together. The projects constitute an environment in which members of user and developer communities collaborate towards a common goal. Hence, these communities act as a kind of equivalent of a biological species in the sense that, like species, they evolve over time, and interact with other communities in their environment.

There are many definitions of software ecosystem, emphasizing different aspects [110]. Basically one can distinguish between ecosystems in-the-large and ecosystems in-the-small [81]. The first view is business-oriented [114, 17]. It adheres to the definition of software ecosystem proposed by Jansen et al. [80]: “a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them”. The second view on software ecosystems “in-the-small”, focuses on the technical and social aspects of an evolving family of software projects and their communities [141]. According to Lungu et al. [106], a software ecosystem is “a collection of software projects which are developed and evolve together in the same environment”. There are many popular software systems that correspond to this notion of software ecosystem. In the context of OSS systems, Apache, Linux, Debian, Ubuntu, GNU, Gnome, and KDE are examples. All of them are made of software applications developed as the pieces of a puzzle in order to provide a set of closely related features. In the context of this dissertation, we extend the Lungu’s definition by taking into account the software projects as well as the communities that surround them.

1.2.3 Social Aspects of Software Ecosystems

In the context of software ecosystems, it is important to study the social aspects of how members of a software development community interact, communicate and collaborate.
Indeed, important changes in the developer community may significantly influence the way in which the software project will continue to evolve over time. Examples of such changes are the unexpected departure of key persons, the takeover of the project by a new developer community, a major change in the way tasks are assigned to persons, changes in the collaborative structure, the appearance of a new competing project, and technical changes (such as moving from the centralised to a distributed version control system, or the introduction of web-based support tools such as Gnome Live\textsuperscript{3} and Damned Lies\textsuperscript{4}).

A better understanding of these social aspects is therefore needed to come up with prediction models of software evolution, guidelines and best practices that allow these communities to improve upon their current practices, and tools that can be used by the community to control and improve upon their current work processes, to communicate more effectively, and to make the software more attractive to its users.

Figure 1.1 schematically represents the socio-technical notion of software ecosystem that captures our interest. We define the ecosystem community as the collection of all individuals that are developing or using a coherent collection of programs (constituting the software system)\textsuperscript{[63]}. This ecosystem community may be subdivided in (possibly overlapping) sub-communities. For example, one can distinguish between the user community, containing all individuals who use an executable version of the software system, and the developer community, containing all individuals who are in charge of maintaining and improving this software system over time. Some individuals may not belong to any of these sub-communities, if they are in charge of other activities that are relevant for the software ecosystem as a whole (such as managing the websites or mailing lists, for example).

### 1.2.4 GNOME

In order to address the research goals we need to select as a case study a software ecosystem with at least the following characteristics:

- it should have a long development history (at least a decade);
- it should possess a large ecosystem community involving many different contributors;
- its contributors should also be active in other activity types besides coding, so that we can analyse how the contributors specialise themselves in these activity types;

\textsuperscript{3}https://live.gnome.org/
\textsuperscript{4}https://l10n.gnome.org/
Figure 1.1: “In-the-small” view on software ecosystems. Dotted lines represent the communications between entities and arrows stand for the relations between types of entities.
1.2. CONTEXT

- it should contain a large number of projects, many of which still being actively maintained today;
- the projects should be open source as it facilitates data extraction, and replication of our research results;
- the ecosystem should be well-known to researchers and open source developers.
- the projects should not be isolated, i.e. there should be some kinds of interaction between the projects of the ecosystem and the developers of these projects.

We have selected the GNOME ecosystem\textsuperscript{5} as a case study because it satisfies all of these requirements. The GNOME community develops a popular and successful free and open source desktop environment for GNU/Linux and UNIX-type operating systems. The GNOME projects are classified in categories that reflect their main purpose. In total, GNOME contains 1428 projects, 770 of which (i.e. 54\%) belong to the archived category, which corresponds to projects that are not maintained anymore, and 2 belong to the deprecated category, which indicates that projects become obsolete or useless and should not be used. These values were computed on March 28, 2011, based on the project list available at http://git.gnome.org/browse/. The number of GNOME projects has increased since this date. Each of the GNOME projects has a corresponding Git distributed version repository containing all information about the evolution history of the project. We only considered a subset of 1,316 GNOME projects (including 691 archived projects) because of the following technical reasons: some of the Git repositories were not available at the time of extraction, some of the extractions did not produce any results, some of the projects did not contain any committers, and the official repositories of some GNOME projects are not part of the official GNOME webpage.

The lifetime of the considered projects varies widely. Some of the GNOME projects (e.g. gnome-disk-utility) have started in 1997 and are still evolving today (corresponding to a lifetime of 16 years), others (e.g. gnome-contacts) were created more recently and were merely a couple of months old at the moment we extracted the data. In addition, many of the GNOME projects (over 900 of them) appeared to have become inactive recently, their latest commit dating from before 2011. This is in particular the case for most (but not all) projects belonging to the archived category.

GNOME started in Augustus 1997. Until the beginning of 2013, it officially involved about 6,000 persons who have sent near for 1,316,000 commits to 1,418 GNOME projects. Nowadays GNOME refers to the following concepts \cite{55}:

\begin{itemize}
  \item {www.gnome.org}
\end{itemize}
• **GNOME**: a software platform gathering software systems and providing them common services, such as file management or communication infrastructure.

• **The GNOME Project**\(^6\): the community of persons and companies developing and using GNOME. It must not be mixed up with *GNOME projects*, the set of software projects supported by GNOME. In the remainder of this dissertation we will use the term ‘GNOME project’ to refer to individual software applications belonging to GNOME and being stored in a dedicated Git repository.

• **The GNOME foundation**\(^7\): a non-profit association that organises GNOME’s activities by providing a democratic process in which the GNOME community has a voice [55]. Its main goal is to promote the use and development of GNOME and to discuss the GNOME’s roadmap and events. The foundation is made of several committees that are responsible of a particular aspect of the GNOME’s evolution:
  
  - Any person that significantly contributed to GNOME can apply for membership. The members are parts the foundation for two years periods that can be renewed.
  
  - The **Board of Directors** is composed of 11 members democratically elected by the Foundation members. The composition of the Board is driven by rules avoiding the dominance of a company on the project’s roadmap. The Board takes public decisions about the current issues and the global orientation of the project.
  
  - The **Advisory Board** is made of corporate organisations (such as IBM and Red Hat) paying for their admissions and two non-profit organisations (namely the Debian Project and the Free Software Foundation).
  
  - The **executive director** is a paid employee of the Foundation who manages the GNOME Foundation and acts as its representative.

Unless mentioned explicitly, in the rest of this thesis GNOME will refer to the software platform.

Even if GNOME was started in 1997, its first major release, 1.0, was released in March 1999. GNOME was then used in the desktop operating system of Red Hat. As GNOME grew and became a viable alternative to existing desktop solutions, more and more companies assigned human resources to its development. Even if it used different software systems, GNOME 2.0 was based on the same concepts as the previous revisions.

\(^6\)http://www.gnome.org/
\(^7\)http://www.gnome.org/foundation/
This revision marks the beginning of a regular release cycle. Indeed, even if each project belonging to GNOME can follow its own roadmap, a new release of GNOME is released every six months, namely in September and March of each year. The GNOME 3.0 series officially started in April 2011. Because GNOME 3.0 aimed to offer a radically new experience to the community, it has caused some controversy among GNOME users. This last major release is still under active development.

Figure 1.2 shows the evolution of some basic metrics for the considered subset of GNOME projects, aggregated by civil years. Only the repositories listed on the official GNOME’s repositories website are taken into account. The number of commits is the number of modifications submitted to a project version repository. The number of active projects counts the projects that received at least one commit during the considered year, while the number of active authors counts the authors that submitted at least one commit during the considered year, considered the merged identities. Finally, the number of active files counts the files that were created, modified or deleted during the considered year.

The number of active commits and the number of active projects have similar evolutionary patterns with a global increase from 1998 to 2009-2010 and a stagnation (or even a beginning of decreasing) after. Other metrics do not present any similarity with number of commits and number of active projects: the number of active authors dramatically increases in 2009. This increase coincides with the start of preliminary work for GNOME 3.0, even if we cannot provide an evidence for a causal relation between the phenomenon. The number of active files dramatically increases in 2001 and suddenly decreases in 2004. According to the GNOME official website, changes made in GNOME during this period are particularly important: 2001 corresponds to the beginning of GNOME 2 and the gradual migration of GNOME projects from GNOME 1 to GNOME 2 was mainly done during the three next years. In September 2004, the GNOME 2.8 release provides a new file and network management; Nautilus, Evolution, and Calendar, three important software projects are included to GNOME or deeply improved. The following releases continue to improve GNOME and to extend its features by introducing smaller changes.

1.3 Structure of this thesis

This dissertation is structured as follows.

Part I introduces in Chapter 1.2 the context in which the dissertation takes part as well as the main addressed notions. Chapter 2 presents the global background of this
CHAPTER 1. INTRODUCTION

dissertation by examining the research literature related to the analysis of social aspects in open source ecosystems. It also introduces the GNOME ecosystem.

Part II is dedicated to the tools we have used and developed and the methodologies we followed during our empirical studies of GNOME. Chapter 3 presents the general application framework we developed in order to automate the analysis of the evolution of social aspects in software ecosystems. Chapter 4 describes our approach to build a model of the community as close as possible to the reality. Chapter 5 describes our method to classify the changes made to project files into different types of activity such as coding and translation.

Part III discusses further empirical studies of the social aspects in OSS. In Chapter 6, we provide evidence that the workload of OSS contributors follows a Pareto law: whatever the considered activity type, it appears that few persons carry out a lot of work, while the majority carries out only a small effort. Chapter 7 presents a study of the variation and specialisation of workload among the contributors modifying the source code of GNOME projects. Based on new workload and involvement metrics, we statistically analyse how workload and involvement of GNOME contributors are correlated.

Part IV concludes by presenting the future work in Chapter 8 and by summarising the contributions of this dissertation in Chapter 9. We present an additional reflexion about the limitations and further work based on the lessons learned in this thesis.

The appendices present extra material used in our empirical studies but being too verbose to be included in the studies themselves.

The chapters are on the whole self-contained, yet the future work of each chapter is discussed in Section 8.

Table 1.1 summarises our publications on which some of the chapters are based.
1.3. STRUCTURE OF THIS THESIS

(a) Number of commits
(b) Number of active projects
(c) Number of active authors
(d) Number of active files

Figure 1.2: Evolution of basic GNOME metrics.
State of the Art

The topics addressed in this dissertation have gained considerable research interest in recent years. We provide here an, inevitably incomplete, list of pointers to related research. The chapter begins by presenting the past research on open source software after which the recent studies on software ecosystems are discussed. The third section of the chapter talks about the historical studies of social aspects in software development and, finally, the last section presents the empirical research dedicated to GNOME, which is the software ecosystem that we have selected for our own research.

Some parts of this chapter come from our previously published book chapter [63].
2.1 Evolution of Open Source Software

In the early 1970s and the years after Lehman and his collaborators studied the evolution of several software systems, including the IBM 360 operating system [96]. They deduced from their observations a set of principles known as the Lehman’s laws. These laws describe the evolution of proprietary software systems over time. Other empirical studies based on the analysis of other proprietary systems tend to confirm their validity to a certain extent for E-type systems [86, 97]. However, the laws are subject to interpretation so the authors encountered difficulties to strongly confirm or refute them.

Due to the abundance of projects for which the entire history of all software artefacts can be freely analysed and the growing popularity of the OSS paradigm even in an industry settings [178, 16], this kind of software has been subject to numerous research studies. Several of them aim to confirm or refute the applicability of Lehman’s laws to open source software systems. Godfrey and Tu [60] found that the Linux system had a superlinear growth between 1994 and 1999. More exactly, the number of lines of code follows a quadratic trend during this period. Even if this observation confirms the Continuing growth law, it tends to contradict some other ones, namely the Increasing complexity, Self regulation and Conservation of familiarity laws. That suggests that the laws cannot be used to correctly predict the evolution of some open source systems. Replication studies by Robles et al. [137] and Herraiz et al. [71] confirm the superlinear growth trend of the Linux kernel. This important growth is especially pronounced in a sub-module of Linux containing the device drivers used by the kernel. Godfrey and Tu explain that drivers can be generally developed independently and so the increasing number of added drivers does not necessary increase the global code complexity. A possible additional reason for this could be that new drivers are created by taking the code of existing drivers and adapting them. The important code reuse implies less development effort.

Paulson et al. [129] studied both open source and proprietary software systems. They provided evidence that (1) OSS fosters more creativity and (2) OSS projects have fewer defects, and these defects are fixed more rapidly than defects in proprietary projects. They found that the Linux kernel has a linear growth. This observation does not contradict the results of Godfrey and Tu who studied the Linux system as a whole while Paulson et al. considered the Linux kernel only.

Other studies reveal the discontinuity in OSS evolution. By studying thirteen OSS projects, Herraiz et al. observed that six of them have superlinear growth, four others experimenting linear growth while the last three projects grow sublinearly [71]. Capiluppi et al. [26, 25, 22] showed that the growth of OSS systems may be irregular and made of consecutive sequences of growth and plateaus. The growth patterns have also been studied
2.2. SOCIAL ASPECTS OF SOFTWARE

by Smith et al. [153, 154]. They examined twenty-five OSS systems and considered their evolution as a sequence of consecutive growth trends and provided evidence that increasing patterns are predominant over non-growth and decreasing patterns. Their models also suggest that developer motivation and the effect of complexity on productivity are major factors in OSS development.

A more complete and detailed presentation of empirical studies of OSS evolution can be found in a survey by Fernandez-Ramil et al. [52].

As the organisational structure of OSS becomes more and more complex, studies related to group of software projects and ecosystems have been carried out. German et al. [58] studied the evolution of R’s ecosystem. This is made of a small kernel and packages extending the features offered by R. Core packages are installed by default and are developed by core developers while user-contributed and popular packages are optionals and all made by R’s users. German et al. discovered some differences between optional and core packages. The size of the user-contributed packages grows superlinearly, which is significantly faster than for the other packages. Additionally, the study reveals that user-contributed and popular packages are more attractive for users so that it takes less time to build a user community around them than around core packages.

2.2 Social Aspects of Software

Good communication is an essential success factor for any software project [18, 44]. This is especially true for OSS projects, for which it is, in most cases, easier to become involved in the development team, implying that the team structure needs to be more flexible in order to accommodate the easy integration of newcomers and to deal with the frequent departure of developers.

Nakakoji et al. [121] distinguish between developers, bug fixers, bug reporters, readers, and passive users. The persons belonging to the two last categories do not introduce changes in source code repositories, bug trackers, and mailing lists, which makes their identification harder. The authors provided an even more refined classification of developer types: peripheral developers, active developers, core members, and project leaders. Figure 2.1 shows their proposed “onion model” that suggests that there are more core members than project leaders, more active developers than core members, more peripheral developers than active ones, etc. Aikainen et al. [5] showed that the contributors community of the Linux kernel is structured as a such model.
In their replication study, Dingh-Trong and Bieman [46] studied similar hypotheses by studying the FreeBSD system. They provided evidence that FreeBSD includes a small set of core developers involved in few parts of the system, and a larger set of top developers implementing 80 percent of the system. Yu and Ramaswamy [183] also made a distinction between core and associate project members, but unlike Nakakoji et al. [121] their approach infers roles automatically by clustering developers based on the frequency of their interaction.

Robles et al. [141] studied the evolution of core teams of developers in open source projects. They visually studied the activity patterns of developers to identify ‘code gods’ projects in which core developers are almost the same during the entire projects’ lifetimes. Poncin et al. [133] proposed a classification of developers involved in aMSN and the bug tracker of the GNU compiler collection project into the classes suggested by Nakakoji et al. [121] and discussed the applicability of classifications rules based on commits done, and mails and bug reports sent. They were only able to consider a subset of classes because the data sources they used do not take into account some of them. For instance, passive users, and readers are not tracked down by the tool used in their study for the reason detailed above. They observed that the organisation of the developers involved in aMSN does not respect the onion model.

Terceiro et al. [41] compared core and peripheral developers in open source projects, and observed that core developers introduce less structural complexity than peripheral developers in general, implying that a stable and healthy core team contributes to the sustainability of open source projects. Capiluppi et al. [23] analysed 400 open source projects, their evolution, and the developer communities responsible for their maintenance. They
2.2. SOCIAL ASPECTS OF SOFTWARE

distinguished between stable and transient developers based on the amount of changes they perform and concluded that most of the studied projects have no real developers community but only few regularly contributing developers. Shibuya and Tamai [151] also distinguish between frequent and occasional contributions, based on the number of commits developers contribute each month. However, even though they distinguish between different **activities** related to involvement in a project (e.g., participating in mailing lists, reporting bugs, developing), they do not distinguish between different **development activities** (e.g., coding, testing, writing documentation). Nevertheless, this distinction is required for determining how the developers involved in a software project organise themselves by specialising their activities and for studying the relations that may exist among these specialists.

Abreu and Premraj [1] studied the correlation between developer communication and software quality. They showed a statistically significant correlation between communication frequency and number of injected bugs in the software. Nagappan et al. [120] studied the impact of the developer organisation on software quality and found evidence that several organisational metrics are effective predictors of failure-proneness.

Mockus et al. [118] carried out a comparative study of Apache and Mozilla to investigate the roles and responsibilities of developers. Using both the source code repository and the mailing list of these projects they highlight that there is a set of implicit conventions among developers that implies an intensive communication. Because the communication is not scalable (one cannot linearly increase the communication intensity without adding more human resources), a strategy is needed to restrain the number and the size of communications. Apache seems to have a very efficient approach that consists of a minimal server core with a well-defined interface.

Madey et al. [109] and Van Antwerp and Madey [7] performed an analysis of the social networks involved in open source software development. They classified developers in clusters, distinguishing persons involved in single-man projects, developers involved in a single project and having no contact with other networks and developers communicating through numerous social networks. They concluded that open source development projects can be modelled as a set of social networks where power laws can be observed at many scales. Martinez-Romo et al. [111] went further and provided a methodology to analyse open source social networks to assess the relation between an open source project community and a company. Studer et al. [161] extended their research by analysing the KDE ecosystem and obtained the same results.

Bird et al. [15] analysed social networks emerging from discussions in Apache mailing lists and observed that the mail transmission follows a Pareto distribution: few mailers sent and received most of the messages while most mailers only sent and received very
few messages. Mailers tend to form a small-world network at several points of view; for instance, few mailers received messages from an important number of persons while most mailers received messages from few senders. A strong correlation between mailing and coding activities have been found and evidence is provided that developers have a social role more important than the other mailers in the mailing lists.

2.3 GNOME

 GNOME, the OSS that we have selected for the empirical studies reported in this dissertation, being a large open-source software ecosystem, is a very popular case study in software evolution research. Many researchers have focused on its evolution.

In 2002, Koch and Schneider [88] presented a study of GNOME based on its CVS repository and available discussion lists. They quantified the effort of GNOME contributors and provided evidence that the number of active contributors is correlated to the effort. The authors discovered that OSS systems involve more contributors than proprietary systems but only a few core contributors are responsible for the overall project. The arrival of new contributors also coincides with an increase of communication among the involved contributors. The communication decreases after the joiners came in.

German [55] was the first to study GNOME as a global system made of subsystems interacting together. He analysed the social organisation of GNOME and tried to understand the underlying practices that favour the success of software projects. He observed that coders are far from being the only contributors to a project. Many valued community members are non-programmers, being involved in other important activities such as documentation and translation. He also observed that some members are paid employees, while others are working on a voluntary basis. His study provides evidence that GNOME infrastructures tend to attract new contributors by offering a set of services to support their efforts.

Casebolt et al. [30] found an inverse relation between file size and the notion of author entropy, suggesting that large files are more likely to have a dominant author than small files. The notion of author entropy characterises the distribution of author contributions to a file, and is therefore related (at least in spirit) to our use of inequality indices such as Gini or Theil. The main difference is that we did not focus on the author collaboration for individual files.

Lungu et al. [107] studied the GNOME ecosystem by means of the Small Project Observatory [106], a software ecosystem visualisation tool they used to study the translation
2.3. GNOME

and coding activities of GNOME contributors. They also considered the evolution of activity in GNOME, quantified as the number of files changed per month. They found that a few contributors were continuously involved in GNOME while a rapid turn-over of contributors was generally observed. A significant number of contributors offered a one-shot contribution or were involved for a short period only. Another observation was that some contributors acted as a set, entering and leaving GNOME in the same time. Lungu et al. detected three main phases in GNOME between 1998 and 2009. Until 2000 GNOME was in an introduction phase during which only few projects have existed with low activity. The growth period succeeded the introduction one, from 2000 to 2003. During this phase two projects, namely Nautilus and Evolution, were intensively developed and concentrated almost all the contributors’ efforts. Finally a maturity phase was observed from 2003 to 2009. The maturity phase is characterised by a cyclic sequence of peaks of activity in January and July due to the release cycle policy discussed in Section 1.2.4.

Robles et al. [139] proposed to divide the ecosystem community into overlapping sub-communities based on the type of activity the members are involved in. Certain types of activity (e.g. coding) require more communication and synchronisation between the involved persons than others (e.g. documentation). They also found that some types of activity (e.g. coding) require significantly more effort and concentration than others (e.g. documentation). To define the activity communities for each considered activity type, the authors analysed the commits a developer has made to the source code repository over a project’s lifetime. To all files belonging to the commit, a global activity type is associated. They propose eight different activity types: documentation, images, localisation, user interface, multimedia, code, build, and development documentation.

In 2010, MSR focused on software ecosystems and, more in particular, on the relationships between packages, by relying on information stored in the SVN version control system and the mailing list archives [74]. There were two contributions to this mining challenge that used GNOME as a case study. Krinke et al. [93] focused on the reuse and cloning of code between the different GNOME projects. Luijten et al. [105] focused on the process and efficiency of issue handling. Their study highlights significant fluctuations in bug solving efficiency over time in GNOME. This efficiency significantly varies from a GNOME project to an other one.

Neu and Lichter [124] proposed Complicity, an alternative to the Small Project Observatory platform that combines visualisations and metrics extraction to analyse the evolution of GNOME. The tool offers several views devoted to different kinds of analysis: (1) the analysis of the ecosystem as a whole or of a specific project; (2) the study of the roles of contributors; (3) the description of a particular contributor’s involvement over time. The role view showed that most of GNOME contributors can be classified either as translators or as coders. The tool highlighted an atypic contributor who was involved
as coder and translator in GNOME for a long time and who sent a number of commits significantly bigger than the other contributors.

Linstead and Baldi [100] mined the GNOME Bugzilla database, and Shihab et al. [152] mined the GNOME Internet Relay Chat (IRC) meeting channels. Schackmann et al. [146] analysed the change request process in GNOME by analysing the explicit and implicit requirements concerning the bug writing, triage and fixing. They divided the persons involved in the change request process into several roles: general users report new change requests (defect reports and improvement requests) while bugsquad members analyse the change requests and confirm or discard them. Developers can submit new change requests and mark them as fixed after they applied the requested change. Quality of change request process is assessed by the job done by each category of involved persons. The authors determined that numerous change requests are redundant or invalid. Most of the change reports are triaged in less than a day and are quickly fixed for good. The study does not provide a general conclusion: some projects in GNOME improved their ability of processing change requests while some other presented a decreasing quality or a lack of observable trend.
Part II

Tooling
In order to realise the empirical studies on the GNOME ecosystem discussed in Part III, we created and used a general application framework that incorporates a database storing all relevant information related to a software ecosystem and that was obtained using several repository mining tools. The framework provides a unified data source to analysis and visualisation tools. One such visualisation tool has been directly integrated in order to get a first quick overview of different aspects of the software project under study.

The framework is extensible in order to accommodate more and different types of input and output, depending on the needs of the user. For instance, the data contained in the framework can be easily be exported to statistical tools such as R to offer numerous statistical analyses.

This chapter presents the framework, compares it against existing solutions, and shows how we can use this framework for carrying out concrete ecosystem evolution experiments. Most content of this chapter is based on an article published in IWPSE 2010 [61] and a book chapter being part of the *Software Ecosystems* book [63].
3.1 Introduction

As explained in Section 1.2.3, we need to focus not only on the source code but also on other elements to get a full picture of an ecosystem’s evolution. A project belonging to an OSS ecosystem typically provides data related to its evolution, thanks to publicly available tools, such as a version control tool, one or more mailing list(s) and a bug tracker. However, this data is generally dispersed among several tools specialised in a particular kind of information: there is no central place where the data related to the project’s history is centralised. In addition, the projects belonging to an ecosystem typically do not have a common data repository bringing together all the aspects of their evolution.

This is in particular the case for GNOME for which each project possesses its own version repository and mailing list. The official GNOME webpage\(^1\) also provides more general mailing lists that may be related to several projects. For instance, there is a mailing list for each officially supported language. GNOME also provides a bug tracker\(^2\) that handles the reported issues and change requests of all GNOME projects.

In order to provide a unified access to information about the developer and user communities that surrounds the ecosystem and to realise the goals we defined in Section 1.1, we developed a generic and extensible framework enabling the empirical study, analysis, visualisation of OSS ecosystems.

3.2 The framework

3.2.1 Overall architecture

The OSS ecosystem analysis framework that we propose builds upon existing data extraction tools such as the ones proposed by the LibreSoft group\(^3\), specialised for the extraction of data from OSS repositories [140]. The framework, schematically shown in Figure 3.1, is built according to a layered architecture.

\(^1\)https://mail.gnome.org/mailman/listinfo
\(^2\)https://bugzilla.gnome.org/
\(^3\)http://libresoft.es/
3.2. THE FRAMEWORK

![Diagram of the framework for extracting, processing and analysing open source software ecosystems. Arrows show the data flows.](image)

**Extraction layer.** The *extraction layer* provides tools to extract useful data and metadata from the various data sources that constitute the software ecosystem. Typically, there will be: multiple version repositories containing the source code commits for each project belonging to the ecosystem; change trackers and bug trackers containing all feature requests and problem reports as well as the resolution process; and mailing list(s) containing all the e-mails exchanged among developers. Depending on the ecosystem, other data sources may be available as well, such as websites, fora, instant messaging channels, and RSS feeds.

**Data post-processing layer.** The extracted information is stored in a persistent database, and further processed by the data post-processing layer. This post-processing resolves inconsistencies found in the data, and enhances the database with additional information obtained by different types of tools. A first type of post-processing is *identity matching*, which serves to associate a unique identity to different accounts and logins used by the same person in the same or different repositories. This process is discussed in Chapter 4. A second type of post-processing is activity type detection, which is used to classify the contributions of each community member according to the type of activity they were involved in (e.g. coding, localisation, documentation). This process is discussed in Chapter 5. A third type of post-processing
is metrics computation to compute numerical values from the data, that measure particular quality attributes (e.g. productivity, effort, maintainability, complexity, and so on). Chapter 7 presents some new metrics defined to study the workload and the specialisation of GNOME’s contributors.

**Application layer.** After post-processing, the augmented data stored in the database can be consulted by a variety of applications belonging to the application layer. For example, reporting tools can be developed to generate documents summarising the data or the computed metrics, statistical analysis can be used to analyse the data (e.g. regression analysis and correlation analysis), data mining tools such as Weka\(^4\) can be integrated to classify or cluster the data, and visualisation tools can be used to output relevant data to the user in a graphical way. Together, the tools in the application layer can be used for a particular purpose, such as to compare ecosystems against each other, to automatically highlight trends and communication patterns in the ecosystem, to predict future evolutions, and so on. The possibilities are endless and may be adapted to the studied properties of software ecosystems. In this dissertation we focus on communication patterns and workload specialisation among contributors involved in open source software ecosystems.

Because each layer performs a specific task and operates on various types of data, we need to use specific tools adapted to each considered context. For example, the framework needs tools to extract artefacts from each kind of source code repository used by considered software ecosystems, tools to extract metrics from source files written in different programming languages, and so on. To achieve this, the framework is a mixture of home-brew, free and commercial tools to reuse as much as possible existing work. Unfortunately, these tools are not always compatible or use different exchange formats. The framework provides the glue to present all the data in a uniform way, thereby dealing with inevitable inconsistencies and redundancies, and reconciling divergent results. In the next sections we will discuss each of the layers of our framework in more detail.

### 3.2.2 Extraction layer

Most of the contemporary OSS developer communities make use of distributed version control systems (as opposed to centralised version control systems) as they facilitate working in a team of geographically dispersed persons, possibly even working in different time zones. Distributed versioning tools allow developers to work offline, to increase the safety

\(^4\)http://www.cs.waikato.ac.nz/ml/weka/
of data due to a higher redundancy, and to enable a wide variety of workflows adapted to open source software development.

One of the most popular distributed version control systems is Git\(^5\). It is the one that we have selected to support in our framework (by using CVSAnaly\(^2\)), but other variants, such as Subversion\(^7\), Mercurial\(^8\) or Bazaar\(^9\), can be easily added as well if the need arises. In Git terminology, we can distinguish between two types of OSS project contributors: **committer** and **author**. The committer is the person that has the right to commit files to the version repository. The author is the person that actually made the changes to the committed files. The reason for this distinction is that, for ease of management, an author does not always have commit rights, implying that his changes need to be committed by a different person. Since in the study of communities we are interested in the actual project contributors, we will consider the information pertaining to the authors as the most relevant.

The mailing lists are a common way for the communities involved in OSS projects (and therefore in OSS ecosystems) to communicate. Each e-mail sent to a mailing list is forwarded to the community members who subscribed to this mailing list. The tools used for providing such a service keep a track of each e-mail exchanged, and the entire history of the discussions related to OSS projects are typically freely accessible. These discussions are generally stored in *mailbox* files.

The bug trackers are another type of communication between community members. These tools allow a contributor or a user of a software project to submit a *bug report* or a request for a new feature. Other contributors may interact, for instance by changing the status of the bug report, in order to fix the bug or the add the requested feature.

The extracted data are placed in a database having a particular structure. Because it is a hard and unnecessary work to reinvent the wheel, our framework exploits as much as possible external tools and existing databases. The FLOSSMetrics project\(^10\) provides a database scheme for the persistence support and populated databases respecting this scheme [70]. It is a popular means to study the evolution of OSS projects [72]. It also provides tools to extract data from source code repositories, mailing lists and bug trackers. These tools support Subversion, CVS and Git thanks to CVSAnaly\(^2\) [142]; mailbox files

---

\(^5\)http://www.git-scm.com  
\(^6\)http://metricsgrimoire.github.com/CVSAnaly/  
\(^7\)http://subversion.apache.org  
\(^8\)http://mercurial.selenic.com  
\(^9\)http://bazaar.canonical.com  
\(^10\)www.flossmetrics.org
thanks to MLStats\textsuperscript{11}, and the Bugzilla\textsuperscript{12}, Jira\textsuperscript{13}, Launchpad\textsuperscript{14}, Allura\textsuperscript{15}, Github\textsuperscript{16} and Sourceforge\textsuperscript{17} bug tracker thanks to Bicho\textsuperscript{18}. These tools are developed as parts of the Metrics Grimoire project\textsuperscript{19}. FLOSSMetrics and Metrics Grimoire can thus be viewed as a parts of our framework.

Figure 3.2 shows the FLOSSMetric-compliant database schema used in the framework to represent the data extracted from source code repositories. The schema is similar to the one of database generated by CVSAnaly2. An extra table called identity\textunderscore merging and drawn in red has been added to the original FlossMetrics schema in order to store the sets of identities belonging to the same contributors that are found by the dedicated post-processing tool described below. Each commit done in the repository is stored in the \textit{scmlog} table, with the committer, the author and message associated to it. The accounts of persons involved in the repository are listed in the \textit{people} table and the \textit{identity\textunderscore merging} table associates all the accounts belonging to the same contributors to a unique identifier called \textit{merged\_id}. The changes applied on files are described in the \textit{actions} table.

In this schema, the boxes represent database tables while the arrows represent relations between tables.

For each extracted project, the database is populated with the entire project change history at file level. For each project commit, the database contains the creation date of the commit, the committer name, the author name, and the files touched (i.e. added, removed, modified, copied or renamed) during the commit. Within a single commit, the same file can only be touched once. In addition, all files belonging to the same commit are always touched by the same author.

### 3.2.3 Data post-processing layer

The data contained in the database may contain inconsistencies and may lack explicit relations between the stored elements of information. For instance, the accounts belonging
3.2. THE FRAMEWORK

Figure 3.2: Enhanced Entity-relationship model [50] of the database for source code repositories of software ecosystems.
to the same contributor are not highlighted so that study the involvement of these persons taking into account this information is not directly possible. The post-processing is made of modules that aim to introduce complementary information in the database in order to fix these issues. Three modules have been developed: an identity comparator and matcher, an activity detector and a metrics calculator. The two first modules are described in detail in Chapters 4 and 5, respectively.

The metrics module provides the necessary infrastructure to specify and compute metrics that can be used to measure particular characteristics about the software product, software process or software community.

For measuring product metrics, we use tools like SLOCCount\textsuperscript{20} to count lines of code and number of files in a wide variety of different programming languages. Other counting tools, such as CLOC\textsuperscript{21} can be used instead. More tools can be integrated for computing more sophisticated software metrics (e.g. dependency metrics, coupling and cohesion metrics, complexity metrics). An example of such a tool is Analizo\textsuperscript{22}.

For measuring social aspects we have implemented our own suite of metrics that includes measures of the effort of developers belonging to a developer community or to one of its activity sub-communities). These metrics are detailed in Chapter 7.

### 3.2.4 Application Layer

The application layer is made of tools consulting the data stored in the framework’s database in order to generate different outputs allowing the users to achieve empirical studies.

R, the widely known statistical tool\textsuperscript{23}, is intensely used in our empirical study described in Chapter 7.

Visualisations of software systems (and particularly their source code) are widely used in the scientific literature [31], as they provide a relatively straightforward abstraction of some of their properties. In order to allow us to gain a first and fast understanding of the evolution of studied software ecosystems, we developed a visualisation tool in

\textsuperscript{20}www.dwheeler.com/sloccount
\textsuperscript{21}http://cloc.sourceforge.net/
\textsuperscript{22}http://analizo.org/
\textsuperscript{23}http://www.r-project.org/
3.2. THE FRAMEWORK

(a) Heatmap (on 1 March 2010) comparing daily and hourly mail activity for Evince. Lighter is more intensive.

(b) Boxplots of daily e-mail activity for Evince.

Figure 3.3: Examples of visualisations available in the framework.

Java and based on JFreeChart\(^\text{24}\), a free visualisation generator. It is able to represent software systems, including their social aspects, as well as their evolution. For instance, Figures 3.3a and 3.3b show an aspect of the activity in the Evince project, a popular document viewer mainly used in GNOME. Figure 3.3a presents a heatmap of Evince’s activity, for a given day against a given hour. We clearly identify a dark area of low activity in the early hours between 2 and 8 AM. The most e-mail activity appears to occur between 10 and 12 AM.

Figure 3.3b allows us to understand how the e-mail activity is dispersed over different days of the week. Each boxplot is based on a data set of 24 values (one for each hour) representing the number of mails sent in that particular hour. The boxplots reveal an important decrease in e-mail activity over the weekend, as can be expected. There is also not a lot of variation in the number of mails sent during the weekend. Finally, without any exception, when analysing all outliers visualised in the boxplot, we find that they correspond to a significantly higher e-mail activity between 10 and 12 AM. For example, the outlier on Tuesday represents the fact that (over the analysed Evince timespan), 44 e-mails were sent between 10 AM and 11 AM.

The visualisation tool we developed for the framework does not aim to offer neither original nor advanced visualisations of the ecosystem or software evolution. Existing and complete visualisation tools and frameworks, introduced in Section 1.1 already exist and have as main goal to help their users to visually represent software systems, and even sometimes their evolution. However, these tools are typically used for studies that do not

\(^{24}\text{http://www.jfree.org/jfreechart}\)
take into account the social aspects of software systems. From the onset our visualisation was designed to provide a first insight on the framework’s database content in order to help the researchers to orientate further investigations.

3.3 Conclusion

This chapter presented the application framework we developed to provide a unified data storage place on which we can base our empirical studies of OSS ecosystems. It was made of existing and self-made tools which help to populate and consult a FLOSSMetric compliant database containing the information related to a software ecosystem and its projects. Post-processing tools take into account and fix (when possible) the inconsistencies occurring while merging several data sources.

The framework contains tools presenting the information stored in the database to help researchers to achieve empirical studies on the considered systems. It can be easily used to visually represent a wide range of metrics relative to OSS ecosystems. Our framework provides a comprehensive, dynamic way to study social evolution patterns in software ecosystems. It is built upon, and takes advantage of existing tools (like FLOSSMetrics and JFreeChart) that have proven their use in the past.

The framework can be easily extended in numerous ways: developing other post-processing tools; making the framework interoperable with more external tools and databases; adding more visualisations; adding the possibility to combine metrics and to compare different ecosystems; providing wizards to help users to choose the best software for them and to help developers to understand the software evolution and how to improve it.
Our framework presented in Chapter 3 allows researchers to extract and analyse data originating from multiple software repositories to understand the historical development of software ecosystems. These repositories typically do not offer a unified view of the contributors involved in the ecosystem they support: each of the repositories represents the contributor by a distinct account. Thus, in order to understand the social interactions involved in the ecosystem’s activities, one needs to determine the identities (e.g. logins or e-mail accounts) in software repositories that represent the same physical person. To achieve this, different identity merge algorithms have been proposed in the past.

This chapter provides an objective comparison of some of these identity merge algorithms, as well as some improvements over them. The results are validated on a selection of large ongoing open source software projects, including two GNOME projects. Most content of this chapter has been previously published in ‘A Comparison of Identity Merging Algorithms for Open Source Software Ecosystems’ in the Science of Computer Programming Journal [64].
4.1 Introduction

As we have seen in Section 2.2, p. 19, a set of collaborative tools including source code repositories, mailing lists and bug trackers are typically used to support Open Source projects. When carrying out empirical studies on the evolution of software systems, the information coming from these different data sources needs to be merged and reconciled [108] in order to offer a consistent and realistic view of the system development. Even if the collaborative tools tend to be used together today, they offer distinct services that can be exploited separately. That’s why they typically do not share their data. In particular, their databases are not shared and the users must log in independently to each of the project’s tools. The users can then own several accounts for the tools used in a project.

The users can also create many pairs of login/password on a single tool. The reason is that the tools generally only request a valid email address when creating a new account, so that a person having two email addresses can easily create two different accounts within the same tool.

Most of the time, the tools are designed to support (an aspect of) the development of single projects. A community wanting to develop many projects must then install many instances of the tools and associate one of them to each of the projects, so that a person involved in many projects must create several accounts even if the projects are developed by the same community. For example, for GNOME, each project has a separate Git repository and separate mailing lists, so contributors wishing to participate to different projects need a different account in each of these projects and need to subscribe to each of the mailing lists of interest.

One of the main challenges in the merging and reconciliation process is to identify the accounts belonging to the same person and to map these accounts to the concerned person. Doing this manually is too error-prone and too time-consuming for applying it to a set of projects. Automated processes are not perfect either, as they may give rise to more false positives and false negatives than manually.

We present existing automated processes identifying and merging the identities and compare their efficiency through a comparative study. The comparison is about the quality of the merging done by the processes. This quality is expressed by the number of false positives and false negatives observed while comparing the obtained merging with a reference merging.
4.2 Terminology

To avoid any confusion in terminology, this section defines the terms that will be used in the remainder of this chapter.

**Contributor:** A person who contributes to a software project.

**Software repository:** A place where the contributors of a software project store different aspects of the project history. We distinguish the source code repository, in which the changes applied on the source code are stored, the mailing lists that store the emails exchanged by the contributors, and the bug trackers that contain the submitted bug reports as well as their entire histories.

**Identity:** The way a contributor identifies himself in a particular software repository. Each *identity* in a repository has a collection of *labels* that is unique within that repository. Depending on the repository and the naming conventions imposed, these labels can take the form of a valid *email address*, a contributor’s *name*, or a *pseudonym* (a.k.a. *nickname*). Within a single repository, the same contributor may have different *identities* (with different labels). For example, he may use two different e-mail addresses to contribute to a mailing list, or he may commit to a code repository using his real name or a pseudonym. In different repositories, the same contributor may have different *identities*. An identity in a given repository always uniquely identifies a single contributor, but if an *identity* with the same collection of labels occurs in a different repository, it may belong to a different contributor.

**Notation:** In the remainder of this dissertation, we will represent identities as pairs containing the labels and the repository to which the identity belongs.

**Example:** In Figure 4.1, contributor John Smith has two code repository identities (*johnny*, *code-repo*) and (*john <js@gmail.com>*, *code-repo*), and one bug tracker identity (*john*, *bug-repo*). John W. Doe has three code repository identities (‘*Doe, John*’, *code-repo*), (*john*, *code-repo*), and (‘*John, Doe*’, *code-repo*), as well as three mail repository identities (*john@doe.org*, *mail-repo*), (*john.doe@gmail.com*, *mail-repo*), and (*john_doe@hotmail.com*, *mail-repo*).

**Label:** A string characterising an identity in a given repository. Depending on the repository’s nature, the label can be a real name, a nickname or an e-mail address. An identity can have one or more label(s). For instance, the contributors involved in a

---

1This is not true in general. In some systems, such as StackExchange (http://stackexchange.com/), contributors use the same identity for each of the repositories. Nevertheless, the organisation described here is representative of most of the open source software systems.
bug tracker are typically identified by a nickname, while the ones who own an account on a source code repository are typically identified by their names and e-mail address, as shown in Figure 4.1. A label can often be split into a series of parts separated by special characters (such as space, comma, the @ symbol, or a dot). If the label is an e-mail address, it can be split into the e-mail prefix that precedes the @ symbol and the e-mail suffix that follows the @ symbol. If the label is a name, it can frequently be split into a series of parts representing the first name(s), middle name(s), and last name(s) of the contributor. To facilitate the analysis, labels are often normalised, by converting capitals into lower case, removing accents and trailing spaces, and so on.

Example: In Figure 4.1, the label of identity (‘John, Doe’, code-repo) can be split, after normalisation, into two parts John and Doe. Similarly, the parts of label (john.doe@gmail.com, mail-repo) become john and doe after normalisation and removal of the e-mail suffix.

Identity merge: A nonempty set of identities that supposedly identify the same contributor. A false positive (type I error) refers to a pair of identities that are incorrectly contained in the same identity merge, because in reality they represent different contributors. A false negative (type II error) refers to a pair of identities that are not in the same identity merge, even though in reality they represent the same contributor.

Example: In Figure 4.1, a correct identity merge for contributor John Smith would be {{john <js@gmail.com>, code-repo}, {jonnhy, code-repo}, {john, bug-repo}}. An incorrect identity merge for the same person would be {{john <js@gmail.com>, code-repo}, {johnny, code-repo}, {'John, Doe', code-repo}}. It contains a false positive due to the presence of {'John, Doe', code-repo} as well as a false negative due to the absence of {john, bug-repo}.

Merge model: A set of identity merges such that each identity is contained in one and only one identity merge. That is, a merge model is a partition of the set of identities. An identity merge algorithm is an algorithm that produces a merge model by analysing a predefined set of repositories. The reference merge model is a merge model that is used as a reference against which to compare merge models that have been obtained by executing an identity merge algorithm.

Example: The lines in Figure 4.1 represent a merge model of 4 different identity merges (1 for each contributor). This merge model represents the ideal case in which an identity merge algorithm would be able to correctly associate each identity to its corresponding contributor. In practice, this is not possible due to the presence of false positives and false negatives, as the analysis of the results in Section 4.5.2 will reveal.
Figure 4.1: Example of contributors and their *identities* and associated *labels* in different repositories. This merge model represents the ideal case in which an identity merge algorithm would be able to correctly associate each identity to its corresponding contributor.
4.3 Methodology

In order to determine how identity merge algorithms differ from one another, one needs to compare the merge models they create on a selection of software projects.

4.3.1 Software project selection

To select the software projects on which to carry out the comparison, we impose the following requirements.

The software should be open source, since this facilitates accessibility of data related to the activity of contributors involved in the software evolution process.

To avoid sensitivity to variation of the identity merge algorithm, the software project community should be sufficiently large, in the order of hundreds of contributors involved. Projects that have a too large number of contributors will be excluded from the selection since it would be too fastidious to create the reference merge model, compute the merge results, and manually double-check the obtained results.

The software must still be maintained and used today, in order to be representative of a system that is still under active development. This also guarantees that the analysed data will not be obsolete or irrelevant.

The software project must have at least three different and freely accessible repositories: a code repository, a mail repository (mailing list) and a bug repository. To facilitate data analysis and comparison, data from these repositories will be extracted by our framework using the Libresoft tools CVSAnaly2, MLStats and Bicho presented in Section 3.2.2. This choice has an impact on the selected software projects, since the repositories should have a format that can be processed by these tools.

More specifically, the code repository must be based on Subversion, CVS or Git, which are the only version control systems supported by CVSAnaly2. In fact, even for these supported code repositories CVSAnaly2 sometimes encounter problems during data extraction. Hence, only code repositories that can be processed by CVSAnaly2 will be eligible. The mail repository must be stored as mbox files, which is the only file format supported by MLStats. Finally, the bug repository must be a bug tracker supported by Bicho.
4.3.2 Data extraction

Depending on its type and characteristics, a repository can provide different types of identity labels. The code repository and mail repository contain two types of labels, representing names and e-mail addresses, respectively. The bug repository contains labels representing a nickname that may or may not bear resemblance to the real name (depending on the naming conventions imposed or suggested by the project community).

The post-processing tool will extract all these labels will be extracted from the framework’s database, and will provide them as input to the identity merge algorithms for computing the merge model.

4.3.3 Reference merge model

In order to assess the quality of each of the considered identity merge algorithms, their output needs to be compared against a reference merge model representing the merge model that an ideal merge algorithm would propose. This reference merge model is built in an iterative way.

The first iteration contains no identity merges and considers each identity to be independent of the others.

In the second iteration, relations between some of the identities are identified on the basis of information found in text files in the last revision of the considered software project. Identities identified as belonging to the same contributors are merged into a new single identity placed in the reference model. The text files used for the identifications are typically created manually by the project’s maintainers and are therefore incomplete: they do not contain a full list of all relations between all identities. We take into consideration information stored in the following files (if existing), stored in the root directory of the code repository: COMMITTERS, MAINTAINERS, AUTHORS, NEWS, and README. These files are semi-structured, no strict formatting convention is imposed. As such, the quality and quantity of valuable data that can be obtained from these files may significantly differ from one software project to another. Mining changelog files has been used in the past already to gain a better understanding of the open source software evolution and communities [24, 182, 32].

In the third iteration, the reference model is improved and completed with new identity merges that can be obtained by manual inspection of the set of identities. This
inspection is realised independently by three different persons that are not involved in the considered software projects, but having experience in open source software communities. The manual completion of a reference merge model is a labour-intensive and error-prone process. Moreover, the reliability of the reference merge model depends on the practices used by the software project community for which the reference model is built: it turns out to be much harder to merge identities for a project allowing poorly structured user accounts than for a project that imposes more or less strict naming conventions to be used for logins and accounts.

To improve the reference models we contacted the communities involved in each selected project, but they were not able to provide us with a complete, exact or objective reference model that could be used as a basis for the comparison. For privacy reasons, formal aggregating tools are not publicly accessible, if they exist, and are difficult to automatically exploit due to their diversity. Contributors involved in a software project do not tend to give away their personal data spontaneously, even if this data can be accessed through publicly available resources.

The information that is easiest to find online is the association between a commit login and an e-mail address. But this information can often be found in an easier way (as we did) by directly questioning the source code repository.

For instance, if the contributors have to use their firstname and lastname as bug tracker logins, these logins will be easily matched with the person’s real names. The ease for newcomers to enter an OSS ecosystem community can also play a role in the quality of the reference model. If people desiring to become involved in an ecosystem project can easily create an account as committer, mailer or bug reporter\(^2\), the project’s repositories will contain a lot of poetic or cryptic labels, such as \texttt{unamnxx}, \texttt{Cpt_Kirks}, \texttt{happysheep} or \texttt{jp.sittingduck+winehq}\(^3\). In contrast, a contributor entering in a very controlled OSS project is more likely to respect the naming conventions adopted by the community at the risk of being rejected from the community. However, more restricted-access projects might miss potential contributors.

Although the reason of this gap between information contained in the repositories and the text files we used in the second iteration for creating a reference merge model is not explicitly expressed, we suppose it is due to the fact that additional contributors have no account in a repository. Their names and e-mail addresses figure in files to acknowledge them but their activities (essentially translation) are committed by another person. Since their involvement cannot be quantified, these additional contributors are simply ignored.

\(^2\)For instance, the GNOME’s mailing lists provide instant access on request.

\(^3\)These logins have been found in the source code repositories of GNOME and Wine projects.
4.3. METHODOLOGY

4.3.4 Algorithm comparison

Identity merge algorithms aim to offer a more precise view on the historical evolution of software projects by identifying the persons having contributed to multiple data sources. Unfortunately there is, as far as we know, no automated means to objectively compare the accuracy of these algorithms. This is needed since we wish to assess which of the proposed algorithms leads to more relevant results, with fewer false positives and false negatives.

To perform such an objective comparison, we developed a tool that takes data from each considered repository as well as a reference merge model as input, runs all algorithms on the data, and compares the obtained identity merge model with the reference model. The “quality” of the identity merge algorithm will thus be assessed by how closely it approximates the results of the reference model.

All of the considered identity merge algorithms are parameterised. Therefore, for each selected projects, the comparison tool runs and computes the result of each merge algorithm for different values of its parameter. The outcome of each run is then compared to the corresponding merge reference model to determine its quality. This quality is determined by looking at all pairs of identities found in the result of the merge algorithm and the reference model. Four possibilities need to be distinguished, as summarised in Table 4.1:

- We have a true negative ($tn$) if the merge algorithm does not propose to place both identities in the same identity merge, and the reference model agrees with this;
- We get a true positive ($tp$) if both the merge algorithm and the reference model agree that both identities need to belong to the same identity merge;
- We obtain a false negative ($fn$) if the merge algorithm does not propose to place both identities in the same identity merge, while the reference model suggests they should belong to the same identity merge;
- Finally, we have a false positive ($fp$) if the merge algorithm proposes to place both identities in the same identity merge, while the reference model says they should belong to different identity merges.

For each considered parameter value of each merge algorithm on each of selected projects, the number of true and false positives and negatives is used to compute recall and precision (a value between 0 and 1). Recall provides an answer to the question ‘What
identity pair belongs to ... | Reference model
| Yes | No |

| Merge algorithm result | Yes | tp | fp |
| No | fn | tn |

Table 4.1: The four possibilities to determine whether a given pair of identities correctly belongs to the same identity merge.

is the percentage of correct identity merges that have been found by the algorithm? ‘ and can be computed using Equation (4.1) [169]. Because an algorithm can very easily achieve a recall of 1 (by determining that all identities need to belong to the same identity merge), we need the precision to take into account the number of returned false positives as well. Its definition is given in Equation (4.2) [169].

\[
\text{Recall} = \frac{tp}{tp + fn} \tag{4.1}
\]

\[
\text{Precision} = \frac{tp}{tp + fp} \tag{4.2}
\]

In order to assess the efficiency of a merge algorithm, recall and precision are considered simultaneously to determine for which parameter the algorithm produces the best results. The F-measure [168] may be used to represent both precision and recall with a single value. Nevertheless, we decided to not use this value, because it can hide an important difference between recall and precision, and because the F-measure can be simply computed on the basis of them.

All variants of each algorithm (one for each parameter value) can be presented in a two-dimensional graph with the recall on the horizontal axis and the precision on the vertical axis (see, e.g. Figure 4.2a). The best variants of an algorithm are the ones for which the points are Pareto-efficient and then belong to the Pareto front [128].

4.4 Identity merge algorithms

There are in the literature numerous approaches for comparing persons in several contexts. For instance, Christen [33] experimentally compared techniques for detecting names belonging to the same person. He applied the approaches he compared on a dataset containing midwives’ records from the state of New South Wales as well as on the COMPLETE
name database [66]. The considered approaches were traditionally used for taking into account misspelled or mispronounced names. In his study, Christen showed that the variety of methods for determining the similarity between names leads to the conclusion that there is no single best technique available and that methods must be adapted to the context for better matching results.

This section presents the different algorithms for similarity detection between identities we studied. These algorithms take the shape of a predicate determining if two identities belong to the same person. Each of the algorithms can be used in a 2-step merge algorithm which

1. Builds a undirected graph in which nodes represents the identities to compare. Two nodes are connected if the identities they represent are considered as similar.

2. Retrieves all the connected components of the built graph. Each of these connected components represents the identities belonging to the same person. This operation can be achieved efficiently by using the algorithm proposed by Hopcroft and Tarjan [77].

While any identity merge algorithm follows this procedure, the auxiliary functions that are used vary from one algorithm to another:

- **normalise**$(\mathbf{w})$ is not directly used in the merging algorithm but the different similar algorithms can exploit it to normalise an identity label. This function takes a character string $\mathbf{w}$ and converts it into a normalised form by removing accents, converting uppercase letters into lowercase, replacing a sequence of whitespace characters by a single space, and removing beginning and ending whitespaces.

- **nld** can be used by the merge algorithms to compute the normalised Levenshtein distance between two identity names [122]. This distance is a metric for measuring the difference between two character sequences.

### 4.4.1 Simple algorithm

As its name suggests, the simplest identity merge algorithm we designed and implemented uses a very simple similar function to determine if two identities should be merged together.
Algorithm 1: `similar` for the Simple algorithm

Input: \(a\), an identity
\(b\), another identity
\(p \in \mathbb{N}\), a threshold

Output: a boolean value specifying if \(a\) and \(b\) are similar (true) or not (false)

1. \(A ← \text{list}();\)
2. \(A ← A + \text{normalisedNames}(a);\)
3. \(A ← A + \text{normalisedPossiblePseudo}(a);\)
4. \(A ← A + \text{normalisedMailPrefixes}(a);\)
5. \(B ← \text{list}();\)
6. \(B ← B + \text{normalisedNames}(b);\)
7. \(B ← B + \text{normalisedPossiblePseudo}(b);\)
8. \(B ← B + \text{normalisedMailPrefixes}(b);\)
9. return \(\exists e \in A \cap B \mid \text{len}(e) \geq p\)

As shown in Algorithm 1, the algorithm first creates lists containing the normalised names, the normalised email prefixes and the normalised potential pseudonyms of the identities. The potential pseudonyms of an identity are character strings that the identity owner may use as pseudonym. These strings are built using variations of the names and the email prefixes he uses. The function `similar` returns true, i.e. it decides that the identities should be merged, if they have at least one normalised label in common so that this normalised label has a number of characters longer or equal to the threshold \(p\) given as parameter to the algorithm. For the e-mail addresses, only the prefixes are used.

Example: In Figure 4.1, let’s suppose \(p = 3\). In order to determine if \((\text{john@doe.org, mail-repo})\) should be merged with \((\text{john <jd@gmail.com>, code-repo})\), the algorithm creates the lists \([\text{john, jd}]\) and \([\text{john}]\). \text{john} belongs to both of the lists and is four characters long, so the identities should be merged together.

### 4.4.2 Bird’s algorithm

Bird [15] suggested another algorithm detailed in Algorithm 2 and specifically designed to detect identities belonging to committers in a code repository and mailers contributing to a mailing list. The algorithm requires as parameter a threshold \(t\) ranging from 0 to 1.

In a preprocessing phase, all identity labels are normalised and cleaned. The cleaning removes punctuation, suffixes, prefixes, technical terms and frequently occurring words
4.4. IDENTITY MERGE ALGORITHMS

(see Appendix A). For identity labels representing e-mail addresses, only the e-mail prefix is considered. After this preprocessing, the label — representing the name of a person — is split into two parts: the first name (first) and the last name (last). To achieve this, Bird recommends the use of whitespaces and commas as separator characters.

In order to determine if a given identity should belong to a given identity merge, the algorithm compares the identity with all identities belonging to the identity merge, and determines whether these are similar. This is the case if at least one of the following conditions is respected:

- Both identities are found in the code repository and have similar name labels $l_1$ and $l_2$. More specifically, the algorithm uses the normalised Levenshtein similarity $l\_similar(l_1, l_2, t)$ depending on Levenshtein threshold $t$ provided as a parameter of the algorithm (Equation 4.3):

$$l\_similar(l_1, l_2, t) = \begin{cases} 
\text{true}, & \text{if } 1 - \frac{\text{LevenshteinDistance}(l_1, l_2)}{\max(\text{size}(l_1), \text{size}(l_2))} \geq t \\
\text{false}, & \text{otherwise}
\end{cases}
$$ (4.3)

**Example:** In Figure 4.1, the identities (johnny, code-rep), and (john, code-rep) will belong to the same identity merge for threshold parameter $t = 0.5$ because

$$l\_similar(\text{"johnny"}, \text{"john"}, 0.5) = \text{true} \text{ since } 1 - \frac{2}{\max(6, 4)} = \frac{2}{3} \geq 0.5$$

- Both identities are found in the code repository and have name labels $l_1$ and $l_2$ composed of first$_1$, last$_1$ and first$_2$, last$_2$, respectively such that first$_1$ and first$_2$ are similar, and last$_1$ and last$_2$ are similar (according to Equation 4.3).

**Example:** In Figure 4.1, using commas and spaces as separators, the identities (John, Doe, code-repo), and (Doe, John, code-repo) belong to the same identity merge because both of them have the same name parts john and doe (corresponding to a similarity value 1, exceeding the threshold 0.5).

- Both identities have e-mail labels containing a similar e-mail prefix of at least 3 characters. The similarity function is defined as before.

**Example:** In Figure 4.1, the identities (john.doe@gmail.com, mail-repo) and (john_doe@hotmail.com, mail-repo) will belong to the same identity merge for parameter $t = 0.5$ because

$$l\_similar(\text{"john.doe"}, \text{"john\_doe"}, 0.5) = \text{true} \text{ since } 1 - \frac{1}{\max(8, 8)} = \frac{7}{8} \geq 0.5$$
• One of the identities has a name label, and the other identity has an e-mail label. The first one has \textit{first} and \textit{last} parts containing at least 2 characters and being included in the label of the other identity.

\textbf{Example:} In Figure 4.1, using comma and spaces as separators, the identities (’Doe, John’, code-rep) and (john.doe@gmail.com, mail-repo) will belong to the same identity merge because the parts \textit{doe} and \textit{john} of the first identity match the parts \textit{john} and \textit{doe} of the second identity.

• One of the identities has a name label having two parts \textit{first} and \textit{last} and the other identity has an e-mail label \textit{e}. \textit{first} and \textit{last} contain at least 2 characters and are included in \textit{e}, except for one part \textit{p}. The first letter of \textit{p} is included in \textit{e}.

\textbf{Example:} In Figure 4.1, the identities (John, Doe, code-repo) and (jdoe@gmail.com, mail-repo) will belong to the same identity merge because normalisation of \textit{John, Doe} is split in two parts, \textit{john} and \textit{doe}. The first letter of the first part (\textit{j}) as well as the entire second part is contained in \textit{jdoe}.

\section*{4.4.3 Bird’s algorithm extended}

Bird’s original algorithm has a number of limitations. First, it incorrectly assumes that a person’s name contains only two parts, the first name and the last name. However, the proposed splitting processes can be extended easily to take into account an arbitrary number of name parts. Second, the algorithm ignores identities found in bug repositories. Therefore, the algorithm is likely to have worse results than an algorithm that does take such identities into account. Because of this, we decided to implement an extension of Bird’s algorithm that considers identities in bug repositories as well. Just like a committer’s identity name in the source code repository, the identity name in a bug tracker repository can be separated into parts.

\section*{4.4.4 Robles’s approach}

Robles et al. \cite{138} suggested a more complex and high-level identity merge approach based on a set of rules. The first rule consists in the use of GPG\textsuperscript{4} key servers to determine coupled e-mail addresses. GPG keys are sometimes used in OSS communities to sign e-mails. We can ask a GPG server, if available, to find the keys of people involved in a particular

\footnote{\textsuperscript{4}GNU Privacy Guard, a free implementation of the OpenPGP standard for public key encryption.}
Algorithm 2: similar for Bird’s algorithm.

**Input**: first, an identity  
second, another identity  
p, a threshold value between 0 and 1  

**Output**: A boolean value specifying if a and b are similar (true) or not (false).

1. $\text{names}_A \leftarrow \text{clean}(\text{normalisedNames}(\text{first}))$;
2. $\text{names}_B \leftarrow \text{clean}(\text{normalisedNames}(\text{second}))$;
3. $\text{mails}_A \leftarrow \text{clean}(\text{normalisedMailPrefixes}(\text{first}))$;
4. $\text{mails}_B \leftarrow \text{clean}(\text{normalisedMailPrefixes}(\text{second}))$;
5. if $\exists a \in \text{names}_A, b \in \text{names}_B$ so that $l\_\text{similar}(a, b, p)$ then
6. return true;
7. if $\exists a \in \text{mails}_A, b \in \text{mails}_B$ so that $l\_\text{similar}(a, b, p)$ then
8. return true;
9. if $\exists a \in \text{names}_A, b \in \text{names}_B$ so that $(\forall pa \in \text{parts}(a) \exists pb \in \text{parts}(b)$ so that $l\_\text{similar}(pa, pb, p)$) and $(\forall pb \in \text{parts}(b) \exists pa \in \text{parts}(a)$ so that $l\_\text{similar}(pa, pb, p)$) then
10. return true;
11. if $\exists a \in \text{names}_A, b \in \text{names}_B$ so that the first letter of one part of a is in b, and,  
for each other part pa of a, there is a part pb of b so that $l\_\text{similar}(pa, pb, p)$ then
12. return true;
13. return false
software project. Each GPG key can be associated to its owner’s e-mail addresses. This association guarantees that two e-mail addresses belong to the same physical person.

Robles also suggests to rely on specific conventions imposed by the software development process to confirm or reject an identity merge. For instance, the KDE project maintains a list of names, logins, and e-mail addresses for each active developer. Using such a list can significantly improve the merge, by avoiding incorrectly merged or separated identities. This solution is not perfect either because it is very project-specific and thus not generally applicable. Moreover, it is not able to cover all involved identities.

We did not include an implementation of Robles’ algorithm in our comparison for different reasons. Firstly, a correct implementation requires significantly more details than what can be found in the article [138]. More importantly, the need to connect to a GPG server would significantly slow down the comparison process, making it unworkable in practice. In addition, many OSS projects do not make use of GPG servers. Thirdly, the selected projects contain files giving an incomplete list of relations between real names, nicknames and e-mails. In addition, each project uses a different textual structure to present this information, making a fully automated generic extraction process very difficult.

Another specificity of Robles’s approach is to match the e-mail prefix of an identity label with a list of likely prefix candidates, based on a person’s name. These prefix candidates can have the form \(firstname.lastname\), \(lastname.firstname\), or a combination of the first name and the first letter of the last name, and so on. This rule is different from Bird’s that only considers a name split in two parts (whereas Robles’s approach allows for an arbitrary number of parts). On the other hand, Robles requires name parts to be separated by a given character, while the only condition Bird imposes the presence of each part in the e-mail prefix. In our improved algorithm described in Section 4.4.5, we decided to implement Robles’ suggestion for matching e-mail prefixes.

### 4.4.5 Improved algorithm

The third algorithm to be included in the comparison of identity merge algorithms combines ideas taken from Bird’s and Robles’s approach, as well as ideas taken from our earlier experience with analysing open source software developer communities [158]. The proposed algorithm requires as parameter a threshold \(t\) ranging from 0 to 1, and takes natively into account code repositories, mail repositories and bug repositories.

Similar to Bird’s algorithm, identity labels are normalised by removing all insignificant words given in Appendix A. Likewise, the thresholded Levenshtein distance is used to determine the similarity of identities. More specifically, two identities are considered similar if at least one of the following criteria is respected:
• One of the identities has a nickname or a name label, and the other identity has an e-mail label. The normalised label of the first identity and the e-mail prefix of the second identity must be similar, according to Equation 4.3 of Bird's algorithm, parameterised by threshold t.

• One of the identities has a nickname or a name label, and the other identity has an e-mail label. Each part of one of the labels has at least 3 characters and is contained in the other label. The characters ‘’, ‘ ‘, ‘+’, ‘.’, ‘-’, and ‘_’ are used as separators to split a label in parts.

• Both of the identities have a nickname or a name label. These labels have at least 3 characters that are similar according to Equation 4.3 parameterised by threshold t.

• One of the identities has a nickname or a name label having at least 3 characters, and the other identity has an e-mail label. The normalised e-mail prefix of the second identity label must be identical to at least one of the potential e-mail prefixes of the first identity. The list of potential e-mail prefixes is created as follows:

  1. Create a list of all normalised parts of the identity label;
  2. For each permutation of these parts, create a string concatenation of the parts separated by one of the characters ‘’, ‘ ‘, ‘+’, ‘.’, ‘-’, or ‘_’. These new strings are the potential e-mail prefixes.

Example: If the first identity is (’Doe, John’, code-repo) and the second identity is (john_doe@hotmail.com, mail-repo), they will be considered similar because john_doe belongs to the list of potential e-mail prefixes containing ’doejohn’, ’johndoe’, ’doe john’, ’john doe’, ’doe+john’, ’john+doe’, ’doe.john’, ’john.doe’, ’doe-john’, ’john-doe’, ’doe_john’, ’john_doe’.

4.4.6 Further Improvements

In this section we briefly present some alternative approaches for identity merging. Because these improvements have been proposed after we achieved the comparison of identity merging algorithms discussed in Section 4.5, we have not taken them into account. This represents a topic of future work.

Poncin et al. [133, 132] presented FRASR, a tool to transform labels into several match objects. For instance, if the label is a real name it forms a match object called raw name according to Poncin's terminology, whereas the normalised version of the name forms a match object called parsed name. Poncin also generates other variations of the name,
combining the first character of a name’s part with the other parts. These combinations are only added as match objects if the concatenation has at least a given number of characters (3 by default). FRASR computes a similarity value \( sim \) and uses this to determine if an identity \( i \) should belong to an identity merge \( iMerge \). The algorithm proceeds as follows:

1. Initialise a variable \( v_{sum} \) to 0.
2. Perform a pairwise comparison of each match object of the identity \( i \) with each match object of all identities belonging to the identity merge \( iMerge \). For each pair of match objects \( (o_{1k}, o_{2k}) \), associate weights \( w_{1k} \) and \( w_{2k} \) to \( o_{1k} \) and \( o_{2k} \) (respectively) based on the types of the considered objects. The weight associated to each object type is a parameter of the algorithm.
3. Compute the product of \( w_{1k} \) and \( w_{2k} \) and add this to \( v_{sum} \).
4. The final similarity value \( sim \) is \( v_{sum} \) divided by the number of comparisons done. If \( numMatch \) is the number of compared objects, \( sim \) can be expressed as follows:

\[
    sim = \frac{\sum_{k=1}^{numMatch} (w_{1k} \cdot w_{2k})}{number \ of \ comparisons}
\]

5. If \( sim \) exceeds a given threshold, the identity \( i \) should belong to the identity merge \( iMerge \).

Kouters et al. [91] proposed an alternative approach based on Latent Semantic Analysis (LSA). This method aims to be more robust and to offer better results than the pre-existing ones. The algorithm can be summarised as follows:

1. Labels sharing a full email address are grouped together.
2. Names are split and normalised.
3. These name parts are transformed in order to gain a more robust solution: digits-only parts are omitted, names are transliterated, etc.
4. First parts of email addresses are split and normalised as well.
5. Resulting character strings are called documents.
4.5. EXPERIMENTAL COMPARISON

6. A \((m \times n)\) matrix \(M\) is filled, where \(m\) is the number character strings and \(n\) is the number of documents. The value of \(M(i, j)\) is 1 if the string \(i\) occurs in the document \(j\). In order to gain a more robust approach, an empty cell \(M(i, j)\) is filled with the value of the normalised Levenshtein similarity between the string \(i\) and the document \(j\) if this value is greater or equal to a given threshold. The normalised Levenshtein similarity between a string and a document is the maximum of the normalised Levenshtein similarities between the considered string and each of the document parts. Finally, the non-empty values are weighted in order to improve the robustness of the algorithm.

7. The pair of documents for which the cosine similarity is greater or equal to a given threshold are grouped together.

This approach is quite similar to the improved algorithm detailed in Section 4.4.5 but contains different filters and transformations that attempt to provide results that are less sensitive to small variations in the considered labels.

4.5 Experimental comparison

On the basis of the project selection criteria of Section 4.3.1, we selected the following software projects on which to carry out the comparison of identity merge algorithms. The characteristics of these projects are summarised in Table 4.2.

- **Evince**\(^5\), a document viewer forming a part of the GNOME project.
- **Brasero**\(^6\), a simple tool to burn CDs and DVDs. It is also a part of the GNOME project.
- **Subversion**\(^7\), a centralised versioning system commonly used in open and close source software developments.

---
\(^5\)http://projects.gnome.org/evince/
\(^6\)http://projects.gnome.org/brasero/
\(^7\)http://subversion.apache.org/
CHAPTER 4. IDENTITY MERGING

<table>
<thead>
<tr>
<th>Project versioning system</th>
<th>Brasero</th>
<th>Evince</th>
<th>Subversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>age (years)</td>
<td>8</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>size (KLOC)</td>
<td>107</td>
<td>580</td>
<td>422</td>
</tr>
<tr>
<td># commits</td>
<td>4,100</td>
<td>4,000</td>
<td>51,529</td>
</tr>
<tr>
<td># mails</td>
<td>460</td>
<td>1,800</td>
<td>24,673</td>
</tr>
<tr>
<td># bugs</td>
<td>250</td>
<td>950</td>
<td>3,493</td>
</tr>
<tr>
<td># commit accounts</td>
<td>206</td>
<td>204</td>
<td>162</td>
</tr>
<tr>
<td># e-mail accounts</td>
<td>102</td>
<td>610</td>
<td>1,690</td>
</tr>
<tr>
<td># bug accounts</td>
<td>421</td>
<td>964</td>
<td>1,340</td>
</tr>
</tbody>
</table>

Table 4.2: Main characteristics of selected software projects. The reported values are those of the last version of each project. Values for May 2011.

4.5.1 Reference merge models

For Evince, the semi-structured files contain developer-related information providing relations between identities of 3 persons. For Brasero, these files provide 45 unique relations between names and e-mail addresses. For Subversion, the files provide the most complete (in terms of quality as well as quantity) set of identity relations with 156 sets containing the real name, the nickname and the e-mail address of involved persons. Note that some of the identities described in the files do not belong to any of the considered repositories. In that case, these identities and their relations are excluded from the reference model.

Based on the reference merge models, we can better estimate the number of persons involved in each selected project. The fact of taking into account the reference merge models reduces the number of persons involved in the evolution by 12% for Brasero, 9% for Evince, and 10% for Subversion.

4.5.2 Algorithm comparison

On each of the selected projects we ran four identity merge algorithms: the simple algorithm (Section 4.4.1), Bird’s algorithm (Section 4.4.2), the extension of Bird’s algorithm (Section 4.4.3) and our own improved algorithm (Section 4.4.5). We did this for a range of different parameter values and computed the precision and recall w.r.t. the reference model.
All of the identity merge algorithms we wish to compare are parameterised. The simple algorithm is parameterised by the minimal length a string must have in order to be considered as being useful for computing its similarity (in this case, equality) with another string. The other considered merge algorithms are parameterised by a threshold value used to determine if a string is similar enough to another based on the Levenshtein distance between these strings.

Figure 4.2 displays, on a two-dimensional graph, this precision and recall for each parameter of each algorithm. This figure shows that the simple algorithm always obtains a high precision. The recall, however, appears to vary (between 0 and a value around 0.8) depending on the value of the algorithm’s parameter.

When comparing Bird with Bird extended, we observe that the extended algorithm that takes into account bug repositories outperforms Bird on the recall scale: it invariably has a higher recall due to the fact that it is able to take additional identities into account. Except for the Brasero project, the Bird and Bird extended algorithm suffer from a low precision, forcing the user to manually split incorrectly merged identities.

Finally, the improved algorithm obtains recall values between 0.6 and 0.82, while the precision of this algorithm varies from 0 to 0.9 or better depending on the algorithm’s parameter.

The plots in Figures 4.2 reveal that the results of an algorithm vary a lot based on the parameter value. Therefore, we analysed for each algorithm the effect of its parameter value on the precision and recall in Figures 4.3, 4.4 and 4.5.

For the simple algorithm the precision always remains close to 1, while the recall decreases with increasing values of the minimal word length parameter. Only for a low parameter value ($\leq 3$) we get an acceptable high recall value.

The improved algorithm has the best recall and precision for high parameter values. In fact, the best precision and recall is obtained with a Levenshtein threshold of 1, corresponding to a perfect match. This simplifies things a lot, as it means that it is not really necessary to compute the Levenshtein distance, leading to a dramatic improvement in time.
Figure 4.2: Precision and recall for each parameter of each algorithm. Simple algorithm is represented by black crosses, Bird algorithm by blue triangles, Bird Extended algorithm by green circles and Improved algorithm by red squares.
Figure 4.3: Variation of precision and recall for the Brasero project. Recall is represented by blue squares; precision is represented by red crosses. The higher the value of precision and recall, the better.
Figure 4.4: Variation of precision and recall for the Evince project. Recall is represented by blue squares; precision is represented by red crosses.
Figure 4.5: Variation of precision and recall for the Subversion project. Recall is represented by blue squares; precision is represented by red crosses.
4.6 Discussion

From the obtained results we can make several observations. The simple algorithm provides good precision, but recall strongly depends on the parameter value. It is possible to find parameter values that perform well in all cases. Bird’s original algorithm does not perform well if bug repository data is to be taken into account. Its precision and recall is too low, regardless of the parameter value used. The extension of Bird’s algorithm performs better, with very high recall values for some parameter values, but the precision is always very weak, so it is not usable in practice. The improved algorithm, borrowing features from Bird, Robles’s approach and the simple algorithm, has a recall that is acceptable (around 0.6) to good (0.7 or better). Precision varies a lot depending on the algorithm’s parameter. Like for Bird’s algorithm, the best precision is reached for a parameter value corresponding to a Levenshtein threshold of 1.

As such, we can conclude that the simple algorithm performs the best, closely followed by the improved algorithm.

Based on this knowledge, we could consider to combine the best features of both algorithms to come to a better solution. A such combined approach, based on the Improved algorithm (discussed in Section 4.4.5) and Poncin’s algorithm (discussed in Section 4.4.6), is used in Chapter 7 to merge the identities present in the source code repositories belonging to the whole GNOME ecosystem.

For the Evince project, Figure 4.2b and 4.4a confirm the good quality of the Simple algorithm which gives very good recall ($\geq 0.75$) and precision ($\geq 0.89$). Our Improved algorithm works fine, too, if its words distance threshold is high enough. If we consider only perfect matches of character sequences, recall and precision are around 0.75.

Bird’s algorithm and its extension provide bad results, once again. Both of them have a near to zero precision. The extended version can provide an acceptable recall for some parameter values. Curiously, the impact of this parameter is not linear: a better recall is obtained using a small ($\leq 0.33$) or a big ($\geq 0.51$) parameter value. In any case, the two variants of Bird’s algorithm are so imprecise that an additional manual check would be needed to remove all false positives.

When comparing the results across the selected software projects we observe that the more “noisy” and complex the project data is, the worse the merge algorithms behave. The complexity of a project depends on the number of lines of code it contains and the number of persons involved in the project (these two metrics being correlated). The noise of a project is defined as the ratio of fanciful account names over the total number of
accounts. For example, the merge algorithms need to consider more merges for Evince than for Brasero, and the merges are globally more subtle since more persons use an altered version of their names. For instance, a (fictitious) person called ‘Robert Smith’ could have as login ‘bobby.smith’, and ‘John Baker’ could use ‘thebaker’ as pseudonym. We observed that, if a project is maintained by a small number of persons, these persons tend to use serious logins, probably because they organise their work using a relatively strict process. When the project is opened to ‘anybody’, there are more fanciful accounts created by persons that only occasionally participate to the project’s evolution. It could be interesting to scientifically study if this behaviour is generally observed in open source projects.

Bird’s algorithm as well as its extension behave differently depending on the project on which they are applied. Taking Brasero as an example, we observe in Figures 4.3c and 4.3d that Bird’s algorithm and its extension have an overall weak precision. Nevertheless, larger values of the Levenshtein threshold (in particular, values that exceed 0.5) lead to a higher precision. For the other two studied projects (Subversion and Evince), Bird’s algorithm and its extension give a bad precision, independently of the Levenshtein threshold used. We suspect that the different behaviour observed for Brasero is due to the relatively strict naming conventions (i.e. less “noise”) used by the involved persons when choosing their login.

In contrast to Bird, the simple and improved algorithms appear to be more robust to noise in the projects: they have roughly the same behaviour across the three studied projects (Figures 4.3, 4.4 and 4.5). The simple algorithm always has a very high precision with increasing values as the algorithm’s parameter value increases. The recall of the simple algorithm is high for small parameter values, and decreases rapidly as the parameter value increases. The best trade-off for ensuring a high precision and a high recall appears to be a parameter value of 3. The improved algorithm has a better precision and recall for increasing values of the Levenshtein threshold. In addition, the precision and recall values stabilise after a given threshold value that depends on the project under study. As we continue to apply our algorithms on more projects, we expect that the simple and improved algorithms will continue to produce good results if the right parameter values are chosen. However, bigger projects may suffer from a weak precision and recall, because a bigger data set implies more noise and more false positives and negatives.

In addition, identity merge algorithms will not work well if the community does not impose some discipline on the name conventions for identity labels used in the different repositories. We tried to apply the algorithms on Wine, another OSS project with a huge community (several thousands of involved persons). Because no name conventions were imposed on the bug repository, the merge algorithms performed very badly, and it was not possible to create a decent reference merge model without considerable effort.
In presence of e-mail addresses, false negatives are sometimes encountered if the last name of a person belongs the e-mail suffix. We illustrated this in Figure 4.1 where John W. Doe has \textit{john@doe.org} as e-mail address. This merge cannot be detected by any of the implemented algorithms. False negatives often occur because different variants of first names are used. For example, Robert may be the same as Bob or Rob or Bobby; William may be the same as Bill; and so on. A smarter algorithm may be able to detect that William Doe and Bill Doe are the same persons. It is doubtful that this approach will work better in practice since, while reducing false negatives, it will increase false positives.

Most of the false positives are due to the fact that logins or e-mail prefixes only contain a first name. Persons with the same first name may accidentally be merged because their e-mail addresses and bug tracker accounts are considered similar. The algorithms can be adapted easily to avoid these unnecessary merges, at the expense of introducing more false negatives.

4.7 Threats to validity

The comparison carried out in this chapter suffers from a number of threats to validity. For those that are inherent to any empirical study involving the use of OSS projects we refer to Ramil et al. [52] and Stol and Babar [159]. We will only discuss the threats to validity that are specific to our experiment here.

4.7.1 Internal validity

The FLOSSMetrics compliant databases that are used by the identity merge algorithms are not perfect. Due to some issues in the data or tools that were used to create the database, the database contents has some encoding problems that needed to be solved manually. Sometimes, email addresses were not valid (e.g. \textit{null.null} or \textit{john.deamon@none}), which is problematic for the algorithms that need to analyse their structure. To avoid errors and a waste of time, we would need to fix the external tools used if needed or automate the database correction process.

The comparison of merging is very time consuming, even if independent computation units allowing a full exploitation of all available processors and cores. However, some parts of the extraction process are particularly slow due to a built-in pause between each extraction step. For instance, in Bicho, a pause is regularly done in order to avoid
4.7. THREATS TO VALIDITY

an unfriendly stress on the bug tracker. The main bottleneck for running the merge algorithms is the computing power. Because this type of algorithm can be easily split in numerous independent computation units, in the future we may resort to massively parallel grid or distributed computing to speed up this process.

The reliability of data is also a potential problem: one can never be sure that the analysed repositories are correct and complete. The less data we have, the less useful the algorithms will be since the lack of information can lead to erroneous merges.

As the repositories are not always centralised, it is hard to ensure that all related data are collected (e.g. there may be a hidden mailing list or unofficial source code repositories). Nowadays, with developer communities being geographically scattered over the world, distributed version control systems such as Git and Bazaar, are becoming more and more widely used. This is the case for GNOME that provides a Git repository for each of its projects. Since distributed version control systems allow a clone repository to be outdated, there is a risk that we do not have a global view of the software code repository. Fortunately, OSS projects generally use a reference repository containing the whole official source code history.

Although the reference merge models were constructed in an iterative way, using different sources of information, they remain partly and inherently subjective. Without any formal and unified data source containing all persons involved in the project, there is simply no way to obtain a “perfect” reference model. There will always exist a non-quantifiable bias that one can only try to minimise.

There is also a threat to implementation validity. We implemented Bird’s algorithm on the basis of a high-level textual description we found in [15]. We cannot guarantee that our implementation exactly matches Bird’s ideas. Our own simple and improved algorithms, as well as the algorithm comparison tool itself, may also be subject to implementation errors. We tried to minimise this threat by validating the tool and the algorithms through extensive unit test suites.

4.7.2 External validity

In our study we objectively compared the quality of existing identity merge algorithms. Any new algorithm can easily be added to our comparison tool.

The set of selected software projects may be biased and not necessarily representative. We cannot guarantee that the obtained results are generalisable to other projects, but the
results we obtained for the studied projects are quite similar, increasing our confidence that it will remain true for other projects as well. To facilitate replication, and to apply our comparison to other projects as well, our comparison tool is publicly available at its dedicated repository\(^8\). Any researchers wishing to reproduce the results we obtained, or to apply it on other projects or other algorithms can access the necessary information.

### 4.8 Conclusion

Empirical research on evolving open source software systems requires the mining of software repositories (such as code repositories, mailing lists and bug repositories). To mine historical data about software systems development from these repositories, identity merge algorithms are needed. These algorithms allow to determine which identities in software repositories represent the same physical person. As such, it becomes possible to take into account the social dimension, by studying not only the evolution of the software artefacts themselves, but also the interaction with and between the persons producing, using, and modifying these artefacts. This is an important prerequisite for studying how software project communities are structured, how they evolve over time, and how this may affect the quality of the project and product.

This chapter presented different identity merge algorithms, applied them on different open source software projects, and compared their precision and recall. We can conclude that, if its parameter value is rightly chosen, the simple algorithm performs the best, closely followed by the improved algorithm. Our hypothesis is that members of developer communities tend to be conservative in the labels they use for their identities, making the simple approach fit well, and not requiring more sophisticated rules.

While some algorithms clearly outperformed others, none of the studied algorithms obtained sufficiently high recall and precision. This is in line with the views of researchers that implemented some of these identity merge algorithms and proclaimed that, even after fine-tuning the algorithm, a manual check is still needed to avoid false positives and negatives. Given this need for a manual post-processing phase, high recall should be valued over high precision, since it is much more difficult to manually detect false negatives than to find false positives [15, 132].

We also bear out Bird’s assumption according to which it’s easier to manually separate wrongly merged identities than manually merge erroneously split identities. It reinforces our opinion concerning criteria predominance: obtaining weak precision is not a problem

\(^8\)https://bitbucket.org/mgoeminne/herdsman
if a manual post-check on the merge model is done, but having a weak recall is much more problematic.

We also analysed how stable the identity merge algorithms are with varying values of their parameters. On the software projects studied, the accuracy of each algorithm may vary a lot depending on the parameter value, but a good choice of the parameter values allows to optimise the precision and recall. We proposed an algorithm that improves upon the state-of-the-art by having a strong recall and being less sensible to parameter variations. Nevertheless, further improvements remain necessary, as well as a validation on more software projects. An other algorithm is proposed in Chapter 7 for merging the identities associated to all the contributors involved in GNOME’s source code repositories. It combines manual and automatic techniques to provide a merge model with less false positives and false negatives.

In order to carry out the empirical study discussed in Chapter 7, we created an other identity merge algorithm in the post-processing layer of our framework that allows us to merge the identities associated to contributors involved in GNOME’s source code repositories. This approach is a combination of existing techniques and takes advantage of lessons learned by the comparison done in this chapter. This new algorithm is described in Chapter 7.
Activity detection

In order to analyse and to understand how contributors communities are involved in OSS ecosystems, several researchers have stressed the importance of distinguishing between the roles played by the ecosystem community members. The source code repositories associated to software projects contain more than only the changes made to the project’s source code. These tools are commonly used to archive any type of changes applied to the projects, including translation and documentation updates, for instance. The different types of activity generally imply the change of different types of files: coding is achieved by changing source code files, program translation is done by adding or updating dedicated translation files, etc. Therefore the activity types of files touched by the project’s contributors relate to the roles they play in their community. Different types of activity (and so different roles played by contributors) may also be observed in mailing lists and bug trackers: some mailers initiate discussions to fix a problem while others only answer their questions; still other persons are specialised in fixing bugs reported by other persons.

The framework presented in Chapter 3 proposes a tool incorporated in its data post-processing layer to identify the activity types of files involved in source code repositories. This chapter describes the approach followed by this activity detection tool. Most of the content of this chapter has been published as part of an article in the Empirical Software Engineering Journal [172] and a chapter published in the Software Ecosystems book [63].
5.1 Introduction

As discussed in Section 2.2, several researchers have stressed the importance of distinguishing between the roles played by the ecosystem community members in order to gain a better insight of this community.

Nakakoji et al. [121] proposed a distinction between bug fixers, bug reporters, and developers, with a further subdivision of the latter into peripheral developers, active developers, core members, and project leaders. The contributor communities surrounding software projects adopt a layered structure similar to an onion, as show in Figure 2.1, page 20. German [55] observed that coders are far from being the only contributors to a project. Many valued community members are non-programmers, being involved in other important activities such as documentation and translation. German also observed that some members are paid employees, while others work on a volunteer basis.

Robles et al. [139] proposed dividing the ecosystem community into overlapping sub-communities based on the type of activity the members are involved in. To define activity communities for each considered activity type, Robles et al. [139] analysed the commits made by contributors to the source code repository over a project’s lifetime. To all files belonging to the commit, a global activity type is associated. They propose eight different activity types: documentation, images, localisation, user interface, multimedia, code, build, and development documentation. They observed that certain types of activity (e.g. coding) require more communication and synchronisation between the involved persons than others (e.g. documentation). They also observed that some types of activity (e.g. coding) require significantly more effort than others (e.g. documentation).

Hindle et al. [73] associated revisions to four different activity types, namely source revisions, test revisions, build revisions, and documentation revisions. They studied the frequency of each of these activity types before and after a release and show that source revisions are the most common revisions. They also observed that test revisions are generally more frequent before a release than after. Often, the projects they studied present numerous inconsistencies and a global behaviour cannot be inferred from their observations.

Our approach is similar to that of Robles et al. [139], who distinguish between different activity types based on the file names and extensions.

In the rest of this chapter, we describe the methodology we followed to create a post-processing tool, part of our application framework that allows us to determine the activity type associated to each file in a source code repository. By using this tool and analysing the persons who commit changes on files, we are able to associate roles to contributors. A study of contributors’ specialisation is described in Chapter 7.
5.2 Methodology

We implemented an enhanced version of Robles’ approach and performed a more fine-grained distinction by working at the level of individual touched files, rather than the entire commits in a project’s version control repository. While Robles et al. proposed 8 different activity types, we expand upon their classification by performing a more fine-grained distinction and by working at the level of individual touched files, rather than the entire commits in a project’s source code repository. The additional activity types we consider are testing, database, library, configuration, meta-data, and unknown.

The type and amount of activity the community members carried out can be estimated on the basis of the types and the number of files that are touched during each commit in the project’s source code repository. To this end, we collect fully qualified file paths, including the directory hierarchies, and the names and extensions of the files that have been touched for each commit in the version control history of each project.

The 14 different activity types that we have defined are summarised in Table 5.1. Because the exact set of activity types can depend on the software ecosystem under study, the types of activity to be detected as well as the rules (expressed as regular expressions) used to detect them can be configured and changed easily with our framework. For each software ecosystem, the rules used to detect activity types may vary, and need to take into account domain-dependent information, such as the particular file naming conventions used by the contributors involved in the source code repositories, conventions used for the directory structure (e.g. all third-party libraries are stored in a /library/ directory), and so on.

**Documentation** (doc) helps the final user in getting acquainted with the application. **image** refers to all picture files used as part of the software project (e.g. button icons, illustration in documentation). **Localisation** (l10n) consists in adapting the software for other cultures, and includes translation activities. **User interface** (ui) is concerned with providing a graphical user interface to interact with the application. Files associated to **multimedia** (media) contain sounds, videos, and other multimedia resources (excluding images, which are categorised separately) that are used in the software. **code files** describe the software logic, whereas **test files** contain the instructions needed to automatically test this logic. Files pertaining to the **meta** activity type are not a direct artefact of the projects, but support the software development process. **Configuration files** (config) are used by developers to describe some project properties, whereas **build files** are used to help the developers and/or users to build a binary from the available resources. The **development documentation** (devdoc) aims to help persons involved in the project’s development to maintain and improve the system. Files attached to the **database** (db) activity
are used by the application as knowledge management resources. *Library* (lib) files contain third-party software. The last activity type, labelled *unknown*, contains all files not contained in any of the previous activity types.

Table 5.1: List of all activity types detected by the framework, by analysing the files committed in the version control repository of a software project.

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Description and example rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Files containing the software logic and code.</td>
</tr>
<tr>
<td></td>
<td>Example: files with extension .c, .h, .cpp relate to programs written in C/C++, files with extension .java relate to Java programs.</td>
</tr>
<tr>
<td>test</td>
<td>Files containing instructions to test the software logic and code.</td>
</tr>
<tr>
<td></td>
<td>Example: Java source code file ending by Test.java, files belonging to a /test/ or /testing/ directory.</td>
</tr>
<tr>
<td>devdoc (development documentation)</td>
<td>Files that help persons involved in the project’s development to maintain and improve the system.</td>
</tr>
<tr>
<td></td>
<td>Example: a file called TODO in the root directory.</td>
</tr>
<tr>
<td>build</td>
<td>Files that are used to help the developers and/or users build a binary from the available resources.</td>
</tr>
<tr>
<td></td>
<td>Example: the Makefile.am file, the INSTALL file in the root directory.</td>
</tr>
<tr>
<td>doc (documentation)</td>
<td>Documentation files that help the end-user with using the application.</td>
</tr>
<tr>
<td></td>
<td>Example: files with extension .man, file names containing doc or docs.</td>
</tr>
<tr>
<td>l10n (localisation)</td>
<td>Files that are used to adapt the software for other cultures, and to translate the software into different spoken languages.</td>
</tr>
<tr>
<td></td>
<td>Example: files with extension .po, files in directory po or in directory locale, and so on.</td>
</tr>
<tr>
<td>image</td>
<td>Image files used as part of the application (e.g., button icons).</td>
</tr>
<tr>
<td></td>
<td>Example: files with extension .jpg.</td>
</tr>
<tr>
<td>media (multimedia)</td>
<td>Files containing sounds, videos, and other multimedia resources (excluding images) that are used in the software.</td>
</tr>
<tr>
<td></td>
<td>Example: files with extension .mid.</td>
</tr>
<tr>
<td>lib</td>
<td>Files that contain third party software.</td>
</tr>
<tr>
<td></td>
<td>Example: files belonging the libs directory.</td>
</tr>
<tr>
<td>db</td>
<td>Files used by the application as data management resources.</td>
</tr>
<tr>
<td></td>
<td>Example: files belonging the databases directory, files with extension .sqlite.</td>
</tr>
</tbody>
</table>

Continued on next page
5.2. METHODOLOGY

Table 5.1 – continued from previous page

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Description and example rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>ui (user interface)</td>
<td>Files concerned with providing a graphical user interface to interact with the application. Example: files belonging the gui directory, files with extension .glade.</td>
</tr>
<tr>
<td>config (configuration)</td>
<td>Files used by developers to describe some project properties. Example: files with extension .properties, files containing the term configuration.</td>
</tr>
<tr>
<td>meta</td>
<td>Files pertaining to meta-data that are not a direct artefact of the projects, but support the software development process. Example: files in .svn or .git directories, .project files.</td>
</tr>
<tr>
<td>unknown</td>
<td>Files that do not contain any of the previous activity types. Example: files with extension .zip, .rar and .tar, if not present in a directory matched by a rule.</td>
</tr>
</tbody>
</table>

In order to associate of an activity type to each file belonging to the source code repository, the tool uses a collection of rules mapping files paths to activity types. The rules are made of a pair \((t, e)\), where \(e\) is a case-insensitive regular expression matching the fully qualified file path, and \(t\) is the corresponding activity type. The regular expressions follow the traditional POSIX Basic Regular Expression syntax [79]. Back slash \(\backslash\) is used as escape character distinguishing between . representing any character and \(\backslash\). representing the character ‘dot’, e.g. \(.*\).cpp \ represents all files with the .cpp extension. Forward slash / is used as a directory separator in file paths, e.g. */doc(s?)/.* represents all files in doc and docs subdirectories.

Examples of rules can be found in Table 5.2. For example, \((\text{code}, .*\backslash.c)\) is a rule specifying that any file with extension .c, regardless of its file path, corresponds to a code activity (since it is a C source code file). A complete set of rules, used in Chapter 7 to identify activity types in GNOME files can be found in Appendix B.

To define the regular expressions, domain knowledge must be used. This knowledge can be programming language specific (for instance, programming languages have traditional extensions for their source code files, such as .java for Java programs), project specific (for instance, a project community can decide to place all the files related to documentation in a /help/ directory), or ecosystem specific (for instance, the leaders of the considered ecosystem can require that all projects belonging to that ecosystem must contain all the
files related to their translation in their root directory). Other examples of commonly used naming conventions are the use of file paths, such as /library/, or parts of file names, such as copyright, to provide an indication of the corresponding activity type (in this case lib and doc, respectively). Because these specificities are generally implicit, a manual inspection of the conventions adopted in the source code repositories must be achieved or completed to create the list of rules.

Some files may be classifiable in more than one category. For instance, one can consider that a unit test file is both a source code file and a test file. In the same vein, a picture file may be considered as image file as well as being part of the documentation. Using a particular rule set, a file may be associated to different activity types if multiple regular expressions are applicable to the file. Nevertheless, we have decided that the tool does not allow such situations and permits only a single activity type to each file. The main motivation of this choice is that the activity detection tool was initially designed to study the specialisation of contributors, as detailed in Chapter 7. In that study, we used a metric for measuring the workload globally or for a given type of activity. In order to guarantee that the sum of workloads related to each activity type is exactly equal to the total workload, the activity types associated to the files must be mutually exclusive. More generally, the association of a file to multiple activity types would pose problems in the definition and aggregation of some metrics, as well as in the statistical analysis and interpretation of some results.

In order to resolve situations where several rules with different activity types match a file path, the rules are treated by the tool as an ordered list. Thus, the last rule that matches the file will be used to classify the file under the associated activity type. For example, let’s say a file /test/ClassTest.java matches the rules (code, *.\java) and (test, */test.*\..*). If the rule for the test activity type is checked after the one

<table>
<thead>
<tr>
<th>Activity type t</th>
<th>Acronym</th>
<th>Regular expression e</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>code</td>
<td>.*.cpp</td>
</tr>
<tr>
<td>Development documentation</td>
<td>devdoc</td>
<td>.<em>\changelog.</em></td>
</tr>
<tr>
<td>Documentation</td>
<td>doc</td>
<td>.<em>\man, .</em>\doc(s?)/.<em>, .</em>\copyright</td>
</tr>
<tr>
<td>Images</td>
<td>img</td>
<td>.*.jpg</td>
</tr>
<tr>
<td>Localisation</td>
<td>l10n</td>
<td>.*\po(~?),</td>
</tr>
<tr>
<td>Multimedia</td>
<td>media</td>
<td>.<em>\media(s?)/.</em>, .\mid</td>
</tr>
</tbody>
</table>

Table 5.2: Excerpt of rules \((t, e)\) that may be used to identify the activity types from the file paths and file names.
for code, /test/ClassTest.java will be associated with a test activity. The ordering is manually defined in order to take the domain knowledge into account.

Files for which none of the regular expressions are applicable are classified as unknown. Examples of such files include (i) container and archive files (having the extension .zip, .rar, etc.), (ii) files having an ambiguous, unusual or no extension, as well as files having no specific name or a non-specific path.

In order to assess the time needed to achieve the classification, all the files that were present in all the source code repositories belonging to the GNOME ecosystem since its beginning to January 2013 have been classified using the tool described in this chapter and the rule set described in Chapter 7. The tool has been executed sequentially 20 times on a computer with a 2.54 GHz Intel Core 2 Duo processor and 4 Go 1067 MHz random access memory, after all the file paths have been stored in RAM. The minimal, maximal and mean running times for a complete classification were 90.70 seconds, 92.24 seconds, and 91.04 seconds, respectively. Given the fact that the used data set contains 7,774,408 distinct file paths, we conclude that that tool is fast enough to be applied on large software ecosystems such as GNOME.

5.3 Conclusion

This chapter presented a tool belonging to the post-processing layer of our framework. It offers to us a means to associate a type of activity to each of the files stored in a source code repository.

The rules used to classify the files can be easily changed in order to adapt the tool to the particularities of the studied software ecosystem. Since the classification is only based on the fully qualified paths of the considered files, this tool is fast enough to be applied on ecosystems involving a large number of files.

As part of this dissertation, the activity type identification tool is used in Chapter 7 for determining in which types of activity a contributor is involved (if any).
Part III

Empirical studies
Evidence for the Pareto principle in Open Source Software Activity

In order to gain a deeper understanding of the evolution of OSS ecosystems and the social interactions that surround them, it is essential to study not only the software artefacts created by the tools supporting their evolution, but also their interplay with the different project members (mainly developers and users) that communicate (e.g., via mailing lists) and collaborate in order to construct and evolve the software.

In this chapter, we study how activity is spread over the different members of some OSS projects, and how this activity distribution evolves over time. We observed that the distribution of activity among the persons involved in the studied projects follows the Pareto principle (which states that 80% of a total workload is done by 20% of the involved workers) and exhibits power law behaviour.

The contents of this chapter is mainly based on a previous publication in the SQM 2011 proceedings [62].
CHAPTER 6. EVIDENCE FOR THE PARETO PRINCIPLE IN OSS ACTIVITY

6.1 Introduction

As explained in Chapter 2.2, several tools (including source code repositories, mailing lists and bug trackers) are commonly used to support the evolution of OSS. In order to gain a deeper understanding of the evolution of OSS ecosystems and the social interactions that surround them, it is essential to study not only the software artefacts created by these tools (e.g. source code, bug reports, and mails), but also their interplay with the different project members (mainly developers and users) that communicate (e.g. via mailing lists) and collaborate (e.g. via version control tools) in order to construct and evolve the software.

In this chapter, we wish to understand how activity is spread over the different members of some OSS projects, and how this activity distribution evolves over time. Our hypothesis is that the distribution of activity follows the Pareto principle, in the sense that there is a small group of key persons that carry out most of the activity, regardless of the type of considered activity. To verify this hypothesis, we carry out an empirical study based on the GQM paradigm [10]. Our study takes into account data from source code repositories, mailing lists and bug trackers of OSS projects. We rely on concepts borrowed from econometrics (the use of measurement in economy), and apply them to the field of OSS evolution. In particular, we apply economic indices that have been introduced for measuring distribution (and inequality) of wealth, and use them to measure the distribution of activity in software development.

The remainder of this chapter is structured as follows. Section 6.2 presents related work. Section 6.3 explains the methodology we followed and defines the metrics that we rely upon. Section 6.4 presents the experimental setup of our empirical study that we have carried out. Section 6.5 presents the results of our analysis of activity distribution in three OSS projects. Section 6.6 discusses the evidence we found for the Pareto principle. Finally, Section 6.8 concludes and Section 6.7 presents the threats to validity we observed.

6.2 Related Work

Some researchers have focused on the study of core teams [141, 115] and the inequality of distribution in software development [171, 174, 148]. Our work distinguishes itself from that of most other researchers involved in mining software repositories [38, 45, 1, 56, 133], who tend to focus on the analysis of the software development artefacts (e.g. source code, bug reports, and so on) and the dependencies between those. Instead, our main interest
6.2. RELATED WORK

go to the *individuals* involved in creating and modifying those artefacts, as well as the interaction and communication between those individuals.

The Pareto principle, in economy, states that close to 20% of the persons get close to 80% of the incomes [128]. Adapted to social activities, the principle means that close to 20% of the persons are responsible of close to 80% of the activity. This principle takes part in a more global situation in which values follow a power law distribution (that is, a situation in which a value varies as a power of another value) [117].

Numerous studies have found evidence for the Pareto principle as well as for a power law distribution in human-related networks [125]. For instance, evidence for a power law distribution has been found in the number of citations in papers [136], the number of sexual partners in human societies [99], and many more.

The Pareto principle, Pareto distribution, and related concepts also occur in software engineering. Hunt and Johnson demonstrated [78] that most of the data available in Sourceforge, a software forge for free/open source software\(^1\), follows a Pareto distribution. In software evolution, Herraiz [69] founds a double Pareto distribution in software size (using different measures of size). Mitzenmacher [116] generalised this observation for file system distributions.

Louridas et al. [104] analysed the dependencies of modules in various software systems, namely the J2SE Software Development Kit, the Perl CPAN packages, shared libraries in Unix distributions, Windows executables and DLLs, FreeBSD ports, as well as TeX, METAFONT, and Ruby environments. This study confirms the ubiquity of power law behaviour as well as the presence of scale-free networks of software components.

Empirical studies of large programs [43, 36] reveal that their internal components are often structured in a network having *small-world* and *scale-free* properties: their internal components are connected each other in such a way that the length of the shortest path between most of pairs of components is *small*, and the degree distribution of the network made of these internal components and their relations follows a power law.

Clauset et al. [35] have analysed twenty-four data sets that have been conjectured to follow a power law distribution. Using a statistical approach based on maximum-likelihood fitting methods, goodness-of-fit tests based on the Kolmogorov–Smirnov statistic, and likelihood ratios, they found that some data sets effectively follow a power law while such a power law does not fit the other data sets.

Scale-free networks have also been observed in object oriented software metrics by Jing et al. [83]. In the four OSS they studied, the distributions of all studied metrics (including weighted coupling between classes, and weighted methods per class) follow a power law.

---

\(^1\)http://sourceforge.net/
6.3 Methodology

6.3.1 GQM paradigm

To gain a deeper understanding of how OSS projects evolve, we follow the Goal-Question-Metric approach [170, 92]. Our main research goal is to understand how the activity is distributed over different types of contributors (namely, the developers and the users) involved in OSS projects. Once we have gained deeper insight in this issue, we will be able to exploit it to provide dedicated tool support to the OSS community. This tool support is intended to become part of the application layer of our framework, which is described in Section 3.2.4.

To reach the aforementioned research goal, we raise the following research questions:

1. Is there a core group of OSS project members (developers and/or users) that are significantly more active than the other members?
2. Is there an overlap between the different types of activity (e.g., committing, mailing, submitting and changing bug reports) the community members contribute to?
3. How does the distribution of activity evolve over time within an OSS community?
4. How does the distribution of activity vary across different OSS projects?

As a third step, we need to select appropriate metrics that will enable us to provide a satisfactory answer to each of the above research questions. For our empirical study, we will make use of basic metrics to compute the activity of OSS project members, and aggregate metrics that allow us to compare these basic metric values across members (to understand how activity is distributed), over time (to understand how they evolve), and across projects (to compare the situation between different OSS projects).

6.3.2 Basic metrics

To obtain the basic metrics of OSS activity, we will extract information from three different types of data sources we have at our disposal: source code repositories, mailing lists, and bug trackers. For each of these data sources, we can define metrics that extract and reflect a particular type of activity:
6.3. METHODOLOGY

- **Development activity**: the activity of developers committing changes to a version repository, measured as number of commits.

- **Mailing activity**: the activity of project members posting messages to a mailing list, measured as number of mails.

- **Bug tracker activity**: the activity of persons interacting with a bug tracker, measured in two different ways: number of new bug report submissions, and number of changes to existing bug reports.

Since we are not only interested in a static view of a particular snapshot of an OSS project at a particular moment in time, we will extract each of the above activity metrics during the entire lifetime of the considered OSS projects.

### 6.3.3 Econometrics

Since several of the research questions require a comparison of the basic metrics (across persons, across projects, and over time), we need aggregate metrics that combine the basic metrics. This is valuable, in particular, if we want to reason about the distribution of activity across OSS project members.

To study such distribution, we borrow ideas from econometrics. This discipline uses statistics and metrics to analyse economic data. As an example, various aggregation measures of statistical dispersion have been proposed (e.g. the Hoover [76], Gini [59, 95, 171], Atkinson [8], and Theil [163, 148] indices) and applied to assess the inequality of the wealth distribution among people, regions, countries, and so on. As opposed to traditional aggregation techniques such as mean [173] or median, inequality indices provide reliable results for highly-skewed distributions. Similarly to such traditional aggregation techniques, inequality indices do not require complex application procedures.

Recently, some of these aggregation measures have been used for analysing evolving software systems. Vasa et al. [171] proposed to use the Gini index as an alternative to traditional software metrics. Serebrenik and van den Brand [148] proposed to use the Theil index instead. Vasilescu et al. [171] showed that the Theil index, the Gini index, and the Atkinson index are highly correlated. Following this emerging trend, we will use three different aggregation measures to study OSS activity distribution. Below we provide the definitions of the three aggregation measures we selected: the Hoover index, the Gini index, and the Theil index. These definitions rely on two auxiliary definitions.
Let $X = \{x_1, \ldots, x_n\}$ be a set of values indexed in ascending order ($\forall i \in 1 \ldots n - 1, x_i \leq x_{i+1}$). The sum of all these values will be called $x_{\text{total}}$ (Equation 6.1). The mean of all values will be called $\bar{x}$ (Equation 6.2).

$$x_{\text{total}} = \sum_{i=1}^{n} x_i$$ (Equation 6.1)

$$\bar{x} = \frac{x_{\text{total}}}{n}$$ (Equation 6.2)

The Hoover index, defined in Equation 6.3, is one of the simplest ways to assess inequality of wealth or income. Its value is the ratio of incomes to take up from the richest part of the population to redistribute to the poorest one so that the incomes become perfectly equal. A Hoover index of 0 represents perfect equality, while a high value represents an important inequality.

$$H(X) = \frac{1}{2} \sum_{i=1}^{n} \left| \frac{x_i}{x_{\text{total}}} - \frac{1}{n} \right|$$ (Equation 6.3)

The Gini index, defined in Equation 6.4, provides a more complex way to assess inequality of income. It is based on the Lorenz curve [103] and ranges between 0 and $1 - \frac{1}{n}$ [4], where $n$ is the number of values being aggregated. The Gini index is the value of the surface area between the diagonal and the curve, divided by the surface area below the perfect line. The Gini index of $n$ values is a value between 0 and $1 - \frac{1}{n}$, while 0 expressing a perfect equity and $1 - \frac{1}{n}$ a perfect inequity. If Figure 6.1 represents the curve of a perfect equality and a non perfect equality, Gini index is represented by the blue area between the Lorenz curve and the diagonal line.

$$G(X) = \frac{2 \cdot \sum_{i=1}^{n} (i \cdot x_i)}{n \cdot x_{\text{total}}} - \frac{n + 1}{n}$$ (Equation 6.4)

Yet another index to assess inequality of income or wealth is the Theil index. It is based on the Shannon entropy [163], and is defined in Equation 6.5. To normalise the Theil

---

Figure 6.1: Visual representation of the Gini index. The image originates from Wikimedia².
index so that it always returns a value between 0 and 1, we can apply the normalisation function $N$ described in Equation 6.6 to it [47].

$$T(X) = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{x_i}{\bar{x}} \cdot \ln \left( \frac{x_i}{\bar{x}} \right) \right)$$  \hspace{1cm} (6.5)$$

$$N : t \rightarrow 1 - e^{-t}$$  \hspace{1cm} (6.6)

6.4 Experimental setup

6.4.1 Implementation

In order to obtain replicable and verifiable results, we use our framework described in Chapter 3 for the following tasks:

1. Collect and store the data related to the studied OSS projects, thanks to tools belonging to the extraction layer.

2. In order to gain a better insight of the contributor’s activity, identify and merge identities belonging to the same contributors thanks to a post-processing tool based on the simple merging algorithm detailed in Section 4.4.1.

3. Compute metrics related to the distribution of contributors’ activities, thanks to a dedicated tool belonging to the post-processing layer.

4. Visualise the evolution of these metrics thanks to the statistical tool R, which takes part in the application layer.

For the specific purposes of this chapter, we added a new tool in the application layer of our framework to compute activity distribution (i.e. the relative activity of each involved contributor), and different variants of activity are supported. In particular, we implemented the three types of activity defined in Section 6.3.2. We also added a tool for computing aggregation indices. Only the Hoover, Gini, and Theil index are currently supported. R, the statistic analysis and visualisation tool can directly exploit the information computed by the activity distribution module and the aggregation index module to produce statistical and visual output representing the inequality indices.
6.4. EXPERIMENTAL SETUP

6.4.2 On the Pareto principle and power law behaviour

Many types of distributions in which people are involved correspond to the so-called Pareto principle: roughly 80% of the effects stem from approximately 20% of the causes [126]. This principle and the associated law have been observed repeatedly in a variety of domains, including software evolution [69, 15].

Answering the first research question of Section 6.3.1 boils down to finding empirical evidence for the Pareto principle in OSS project activity distribution. One should note the difference between the Pareto principle and the related notion of Pareto distribution [68]. While a Pareto distribution satisfies the Pareto principle, the inverse is not true: a statistical distribution may satisfy the Pareto principle without being a Pareto distribution. In fact, many types of power law probability distributions have been observed when analysing human activity, and OSS project activity in particular, and the Pareto distribution is only one them [126, 35]. Many power law distributions satisfy the Pareto principle without being a Pareto distribution.

The second research question of Section 6.3.1 corresponds to determining whether the Pareto principle is present throughout the entire life of the project, and whether it emerges, stabilises or disappears over time.

6.4.3 Selected projects

We analyse the distribution and evolution of activity on the following OSS projects: Brasero³, Wine⁴, and Evince⁵. They have been selected based on a variety of factors: popularity, age, size, availability of the necessary data sources for analysis, and so on. Some of the characteristics of the three selected projects are presented in Table 6.1. In order to gain a better insight of the evolution of activity in a whole OSS ecosystem, the source code repositories of all projects belonging to GNOME are also studied. This is an addition to the results presented in the article [62] on which this chapter is based. Because not all GNOME projects use mailing lists and bug trackers in an important way, the single activity metric we analyse for GNOME is the commit activity.

³http://projects.gnome.org/brasero
⁴http://www.winehq.org
⁵http://projects.gnome.org/evince
CHAPTER 6. EVIDENCE FOR THE PARETO PRINCIPLE IN OSS ACTIVITY

<table>
<thead>
<tr>
<th>OSS project</th>
<th>Brasero</th>
<th>Evince</th>
<th>Wine</th>
<th>GNOME</th>
</tr>
</thead>
<tbody>
<tr>
<td>main programming language</td>
<td>C</td>
<td>C/C++</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>versioning system</td>
<td>git</td>
<td>svn</td>
<td>git</td>
<td>git</td>
</tr>
<tr>
<td>age (in years)</td>
<td>8</td>
<td>11</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>size (in KLOC)</td>
<td>107</td>
<td>580</td>
<td>2,001</td>
<td>7,933</td>
</tr>
<tr>
<td># of commits</td>
<td>4,100</td>
<td>4,000</td>
<td>74,500</td>
<td>1,169,509</td>
</tr>
<tr>
<td># of mails</td>
<td>460</td>
<td>1,800</td>
<td>14,000</td>
<td>N/A</td>
</tr>
<tr>
<td># bug reports</td>
<td>250</td>
<td>950</td>
<td>3,300</td>
<td>N/A</td>
</tr>
<tr>
<td># commit authors</td>
<td>206</td>
<td>204</td>
<td>1,229</td>
<td>5,155</td>
</tr>
<tr>
<td># mailers</td>
<td>102</td>
<td>610</td>
<td>6,879</td>
<td>N/A</td>
</tr>
<tr>
<td># bug reporters</td>
<td>386</td>
<td>961</td>
<td>2,676</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 6.1: Main characteristics of selected OSS projects. The reported values have been computed for the last extracted version before submission of [62], November 2010.

6.5 Empirical study

6.5.1 Activity distribution

Figure 6.2 shows the cumulative activity distribution for the entire history of the Brasero, Evince, Wine, and GNOME communities, and for three types of activity: commits made to the version repository, mails sent to the mailing list, and created and changed bug reports in the bug tracker. These distributions are shown for the last versions of the three projects and the ecosystem we analysed. For the preceding versions, with the exception of the earliest versions, we get similar results.

These distributions illustrate that there always is a small core team of contributors that account for most of the activity.

We find clear support for the Pareto principle in the Brasero project: 20% of the most active committers contribute to about 85% of all commits. For the mail (resp. bug report change) activity, 20% of the most active committers contribute with about 75% of all mails (resp. bug report changes). In this project, for the commit activity, 3 out of 193 persons carry out about 70% of the total number of commits. Even more striking is the fact that a single developer accounts for 60% of the total number of commits. For the mail activity, 7 out of 92 persons sent about 60% of all the mails. For the bug report change activity in the same project, 5 out of 253 persons carry out about 40% of all bug report changes.
Figure 6.2: Cumulative views of distribution of activity for Brasero, Evince, Wine, and GNOME. Brasero is represented by continuous red lines; Evince is represented by dashed blue lines; Wine is represented by dotted black lines; GNOME is represented by a dash-dotted green line. The x-axis shows the cumulative percentage of active contributors, ordered from most to least active. The y-axis shows the cumulative percentage of activity.
Evince has two top committers, each accounting for 15% of the total commit activity. The Pareto principle is also clearly present in this project: 20% of all committers contribute to 80% of the total commit activity, 20% of all mailers contribute to 70% of the total mail activity, and 20% of all bug report changers contribute to 88% of the total bug report change activity. Perhaps an important difference, compared to Brasero, is that the development activity is here a bit more equally distributed (where we found a single person responsible for 60% of the total commit activity).

A study of the Wine community reveals that the number of persons involved is much bigger than for the other two systems studied. This was already apparent from Table 6.1. Figure 6.2 shows the cumulative activity distribution for Wine committers, mailers and bug report changers. The most active committer accounts for 13% of the total commit activity, the two most active bug report changers each account for 13% and 11% of the total change activity, respectively. Concerning the mail activity of Wine, we observed that there is one huge mailer wineforum-user@winehq.org accounting for 48% of the total project mail activity. It turned out that this mailer was in fact an automated transcription of the discussion forum in the mail system. After excluding this outlier from the mail activity data set, we found that the most active mailer only accounts for 4% of the total mail activity. Figure 6.2 also provides evidence for the Pareto principle in Wine. 11% of all committers account for a total of 90% of commits. Similarly, a total of 13% of bug report changers account for 80% of all changes.

The whole GNOME ecosystem has a cumulative commit activity that exhibits the Pareto principle similar to the ones of the three studied projects.

The results of regression analyses, shown in Table 6.2 and achieved thanks to the nls function [11] provided by the R tool, reveal that power laws fit pretty well the individual activity workloads for each of the considered projects and activities. In this table, $\alpha$ and $C$ are values for the parameters of functions of the form $C x^{-\alpha}$ that offer the best fitting regression models. To measure the goodness of fit we use two indicators: RSE and MMRE.

Let $X$ be the set of $n$ contributors and $x_i$, $i \in [1, \ldots, n]$ one of these contributors. Let $y(x_i)$ the actual workload of $x_i$ and $f(x_i)$ the workload of $x_i$ predicted by the regression model. The residuals $\hat{\epsilon}_i$ are the differences between the predicted values and the actual values, as described in Equation 6.7.

$$\hat{\epsilon}_i = y(x_i) - f(x_i)$$  \hspace{1cm} (6.7)

Supposing that the residuals are individually described by the same normal distribution with mean 0 and standard deviation $\sigma$, the residual standard error (RSE) is an
### 6.5. EMPIRICAL STUDY

<table>
<thead>
<tr>
<th>Software system</th>
<th>Activity type</th>
<th>$\alpha$</th>
<th>C</th>
<th>MMRE</th>
<th>RSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNOME</td>
<td>Commit</td>
<td>0.6055</td>
<td>5.767e-03</td>
<td>0.8763</td>
<td>0.0147</td>
</tr>
<tr>
<td></td>
<td>Mail</td>
<td>0.8851</td>
<td>0.0214</td>
<td>0.8537</td>
<td>0.0473</td>
</tr>
<tr>
<td></td>
<td>Bug s.</td>
<td>0.6459</td>
<td>0.0337</td>
<td>0.2946</td>
<td>0.0270</td>
</tr>
<tr>
<td></td>
<td>Bug ch.</td>
<td>1.11645</td>
<td>1.999e-03</td>
<td>0.2740</td>
<td>0.0083</td>
</tr>
<tr>
<td>Brasero</td>
<td>Commit</td>
<td>3.130</td>
<td>7.004e-08</td>
<td>0.9760</td>
<td>0.0039</td>
</tr>
<tr>
<td></td>
<td>Mail</td>
<td>0.8851</td>
<td>0.0214</td>
<td>0.8537</td>
<td>0.0473</td>
</tr>
<tr>
<td></td>
<td>Bug s.</td>
<td>0.6459</td>
<td>0.0337</td>
<td>0.2946</td>
<td>0.0270</td>
</tr>
<tr>
<td></td>
<td>Bug ch.</td>
<td>1.11645</td>
<td>1.999e-03</td>
<td>0.2740</td>
<td>0.0083</td>
</tr>
<tr>
<td>Evince</td>
<td>Commit</td>
<td>0.9385</td>
<td>8.065e-03</td>
<td>1.4786</td>
<td>0.038</td>
</tr>
<tr>
<td></td>
<td>Mail</td>
<td>1.003</td>
<td>1.723e-03</td>
<td>0.3139</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>Bug s.</td>
<td>0.585</td>
<td>0.0265</td>
<td>0.2160</td>
<td>0.018</td>
</tr>
<tr>
<td></td>
<td>Bug ch.</td>
<td>1.093</td>
<td>1.067e-03</td>
<td>0.7552</td>
<td>7.565e-03</td>
</tr>
<tr>
<td>Wine</td>
<td>Commit</td>
<td>0.9021</td>
<td>1.512e-03</td>
<td>9.9933</td>
<td>9.041e-03</td>
</tr>
<tr>
<td></td>
<td>Mail</td>
<td>3.558</td>
<td>2.206e-14</td>
<td>0.9995</td>
<td>9.002e-04</td>
</tr>
<tr>
<td></td>
<td>Bug s.</td>
<td>0.6131</td>
<td>0.0129</td>
<td>0.2169</td>
<td>0.0128</td>
</tr>
<tr>
<td></td>
<td>Bug ch.</td>
<td>0.9665</td>
<td>9.782e-04</td>
<td>1.0151</td>
<td>0.0152</td>
</tr>
</tbody>
</table>

Table 6.2: Best parameters for functions $y = Cx^{-\alpha}$ fitting activity workloads. The fitted data is normalised and ordered by decreasing workload.

The estimate of $\sigma$. Equation 6.8 presents the RSE formula, depending on $p$ the number of parameters in the regression function.

$$RSE(X) = \sqrt{\frac{\sum_{i \in [1, \ldots, n]} \hat{\varepsilon}_i^2}{n-p}}$$  \hspace{1cm} (6.8)

MMRE is the mean magnitude of relative error. The magnitude of relative error of $x_i$, $MRE(i)$, is defined by Equation 6.9 and the MMRE of $X$ is defined by Equation 6.10.

$$MRE(i) = \frac{|\hat{\varepsilon}_i|}{|g(i)|}$$  \hspace{1cm} (6.9)

$$MMRE(X) = \frac{\sum_{i=1}^{n} MRE(i)}{n}$$  \hspace{1cm} (6.10)

Figure 6.3 shows the distribution of commit activity among the projects belonging to the GNOME ecosystem. In the figure, the colour at position $(x, y)$ represents the number of projects belonging to GNOME for which the $x$ most active commit contributors are
responsible for at least \( y \) of all commits done in the project. So the colour in \((0.2, 0.8)\) represents the number of projects in GNOME that respect the 20%-80% rule of Pareto. In this instance, 675 out of 1,327 projects, i.e. 50.9% of them, respect the rule or have an even more unbalanced workload distribution.

Figure 6.3: Number of projects belonging to the GNOME ecosystem that respect a given distribution of commit workload. The x-axis represents the relative number of most active contributors; the y-axis represents the relative number of commits done. The colour in \((x, y)\) represents the number of projects in GNOME so that at most \( x \) contributors have sent at least \( y \) commits in the source code repositories associated to the projects.

6.5.2 Activity overlap

To answer research question 2 of Section 6.3.1, we determined the overlap between categories of activities (committing, mailing, changing bug reports), by analysing those persons that were involved in different activities. Figure 6.4 presents the results of this
6.5. **EMPIRICAL STUDY**

analysis. The contributors (i.e., those contributors owning the 20 accounts associated to most of the commits) in source code repositories, mailing lists, and bug trackers of Brasero, Evince, and Wine are placed in Venn diagrams, as Figure 6.4 shows. Even if the Venn diagrams of Brasero and Evince projects don’t show an important overlap in terms of contributors, we have established that the contributors involved in more than one activity are responsible of most of the changes occurring in the activities in which they are involved.

![Venn Diagrams](image)

Figure 6.4: Overlaps of the top 20 most active contributors of the Brasero, Evince, and Wine communities for the three considered categories of activity (November 2010).
More in-depth analyses reveal that, in Brasero, the two most active committers (out of the top 20) are also very active mailers and bug report changers. In fact, the same two persons account for 67% of the top 20 commit activity, 34% of the top 20 mail activity and 27% of the top 20 bug report change activity in Brasero. We also observe that three of the 20 most active mailers are also active as top 20 bug report changers.

In Evince, the 4 most active persons of the community contribute to each of the three considered activity categories. Together, they account for 35% of all top 20 commits, 26% of all top 20 mails, and 36% of all top 20 bug report changes.

For Wine, we merged two different committer accounts corresponding to the same individual into a single identity (explaining a total sum of 19 instead of 20 for committers). The 4 most active contributors involved in both commit sending and bug reports change are responsible of 8.9% of all the commits sent and 15.9% of all the bug reports changes; the two most active contributors involved in mails sending and bug reports change are responsible of 5.8% of the mails sent and 15.7% of the bug reports changes. In contrast to the previously analysed projects, none of the most active persons were involved in all three different activity categories. Another major difference was that the core group of active persons for Wine was significantly bigger, explaining the smaller percentages we obtained for the most active persons involved in a particular activity.

### 6.5.3 Evolution over time

To find out how the distribution of activity evolves over time, we used the econometric aggregation measures introduced in Section 6.3.3. Figure 6.5 displays the evolution of three indices (Hoover, Gini, and normalised Theil) for the commits in Brasero, Evince, Wine, and the entire GNOME ecosystem. Each data point in this figure corresponds to a different distribution such as the ones shown in Figure 6.2. We observe that, regardless of the index used, the values do not fluctuate a lot, and tend to stabilise over time. For Gini, for example, we see that the index remains most of the time between 0.8 and 0.9, indicating a very unequal distribution of commit activity for all observed versions. This corroborates what we already observed before: a low number of individuals contribute most of the commits. Note that, for GNOME, the Gini index has not been computed for the last years, because the used R function did not allow to compute the index excessively high integer values during computation.

Figure 6.5 shows the evolution of the considered aggregation indices over time, taking into account the entire history of the four considered systems. Vasilescu et al. [171] established that, as we observed in Figure 6.5, there is a strong correlation between Gini,
Figure 6.5: Comparison of three aggregation indices, Gini (blue straight line), normalised Theil (black dotted line) and Hoover (red dashed line), applied to the evolution of commit activity for Brasero, Evince, and Wine projects as well as GNOME ecosystem.
Theil, and Hoover indices. Therefore, in the remainder of this chapter, we will only present the results obtained with the normalised Theil index, as the results for the other aggregation measures will be quite similar.

Figure 6.6 presents the evolution of workload distributions for commit, mail and bug report change activity in Brasero, Evince, and Wine, using the normalised Theil index. In each sub-figures, the lines may begin and end at different times because the histories stored in the considered repositories may cover different periods.

Figure 6.6a shows results corresponding to what we observed in the distributions of Figure 6.2. In all cases, the activity is unequally distributed across individuals. This is especially the case for the commits (with a single committer accounting for 60% of the total number of commits), explaining the high value of the normalised Theil index. For mail activity and bug report change activity, there is also an unequal distribution, but less flagrant than for the commits. This explains why their normalised Theil index curve is below the one for the commit activity.

As Figure 6.6b shows, for each of the activities in Evince, we observe that the normalised Theil stabilises rapidly to a more or less constant value, indicating that the way in which the community distributes its workload is fairly stable over time (even if the persons who contribute to most of the workload may change over time). For commits and mails, the high normalised Theil index indicates that the activity is not equally distributed over the community members. For mails, the normalised Theil index is lower so this activity is more spread over different persons.

Figure 6.6c displays, for mail activity in Wine, a normalised Theil index that is initially much lower than for the commit activity, but after a while its value starts to increase to comparable values. This high value is largely explained by the presence of a single artificial mailer (corresponding to the wine forum). For the bug report change activity, we also observe an increasing growth of the normalised Theil index, revealing an increasing inequality of activity distribution over time.

6.6 Discussion

The results shown in the previous section provide strong evidence for the Pareto principle. The activity of contributors to open source projects is not equally distributed: in all three studied projects, a core group of persons appears to carry out the majority of the work. This kind of behaviour may be related to the way in which open source developer
Figure 6.6: Evolution of workload distributions in Brasero, Evince, Wine, and GNOME for the commits sent (blue straight line), bug report changes (black dotted line), and mails sent (red dashed line).
communities are structured. According to Nakakoji et al. [121] and Antikainen et al. [5], a
typical organisational structure is the so-called onion model. It is followed, among others,
by the community in charge of the Linux kernel. In such a layered model, there is a
single responsible of the project, surrounded by a small core team of software developers,
around which there is a bigger layer of active developers, followed an even bigger layer
of occasional developers. A final layer constitutes those users of the project that do not
contribute anything themselves. The more to the centre of the onion, the more active
a developer, and conversely. Although many variants of this layered model exist, the
general idea behind it remains the same.

Our empirical analysis showed the usefulness of applying results from econometry to
the analysis of the activity in software ecosystems. We used three different aggregation
indices of statistical dispersion, the Hoover index, Theil index, and Gini index. They all
gave similar results, i.e. they tend to evolve in the same way for a given OSS project’s
history. This probably implies that one can freely choose any of these indices to assess the
evolution of activity distribution. This corresponds to the findings of Vasilescu et al. [171]
that compared different aggregation measures applied to software metrics and observed a
strong correlation between them.

For all three OSS projects we studied, the distribution of activity was initially more
equally distributed, but over time the activity tends to become concentrated in a core
group of persons that is significantly more active than the others. This knowledge is quite
important, as the sudden disappearance of some members of the core group may have an
important impact on the future of the software project. In other ways, we found empirical
evidence of the so-called bus factor, the total number of key persons that would, if they
were to be hit by a bus, could lead the project into serious problems. Note that this was
less the case for Wine, by far the biggest of the three projects, where the activity was
more equally distributed over the most active committers than for the other two projects.

The GNOME ecosystem, if considered as a whole, also presents strong evidence for
the Pareto principle, when considering the source code repositories associated to its soft-
ware projects. The unbalance of its workload distribution is similar to the one observed
in Wine, with near to 10% of commit contributors responsible of near to 90% of com-
mits (Figure 6.2a). Considering each of the GNOME projects individually confirms the
existence of this unbalance: in half of them, at most 20% of the most active commit
contributors have sent at least 80% of the commits.

For all considered types of activities, we found a long tail of persons whose activity
rate can be largely neglected: during the entire lifetime of the project, they contributed
once or twice to one of the considered project activity categories (commits, mails and
bug reporting). We observed that the use of logins and accounts in Wine was poorly
structured. For example, many committers have 4 or 5 email addresses that are rarely used. This may be explained by the fact that the Wine community is very open: a simple patch submission is generally sufficient to become an official committer so it is very easy for new persons to become active in this community. We also observed that, except for Wine, the most active project members take part in all these activities. We are under the impression that this behaviour is more apparent for well-structured projects such as Evince than for less structured projects such as Brasero and Wine. Nevertheless we do not provide any strong evidence of this trend.

6.7 Threats to validity

We observed that the use of labels in Wine was poorly structured. For example, many committers have 4 or 5 email addresses that are rarely used. This may be explained by the fact that the Wine community is very open: a simple patch submission is generally sufficient to become an official committer so it is very easy for new persons to become active in this community. This lack of structure is also marked by a higher relative number of unusual labels, compared to the other studied systems: more often than in the other projects, Wine contributors use aliases that cannot be easily associated to each other. As discussed in Chapter 4, a potential threat to validity is that, during our experiments, we encountered some problems to determine which persons contribute to different activity categories (Figure 6.4). The identity merging is a (partially) manual and error-prone process that took a lot of effort. To achieve this, an efficient identity merging algorithm is needed that allows one to identify matches between entities participating in different data sources. But for poorly structured projects such as Wine this may still be quite problematic and difficult to automate.

We used algorithms for detecting and merging the identities that belong to the same person. These algorithms present several threats to validity that are discussed in Section 4.7.

Figures 6.5 and 6.6 present cumulative views of the OSS evolution: the metrics are computed taking into account all the events that occurred before the considered time. Such views can lead to some incorrect interpretations of the trend analysis. In particular, a linear trend may be detected due the fact that old values are always taken into account, making variations less important. Cumulative views take into account all the events that occurred before each considered point in time, and some of them may not be relevant anymore. For instance, a valuable question is to know how relevant is the activity of contributors that have left the considered system a long time ago. A way to resolve this
issue is to use non-cumulative visualisations, by ignoring the events that occurred \textit{a long time ago} or by decreasing the relative importance of \textit{older} events (e.g. either by using a weight function that decreases with the age of the event, or by ignoring old events altogether). Nevertheless, this solution raises new discussions about the way old events must be ignored (for example, concerning the time threshold or the decreasing functions to use).

A property of the aggregation indices that have been computed is that their maximal value depends on the number of contributors of whom the activity is measured. In Figures 6.5 and 6.6 we ignored the contributors that have not yet contributed to the considered systems. Therefore the number of contributors taken into account increases over time and the values of a Theil index cannot be directly compared. The effect of the size increasing is limited by the fact that the four studied systems quickly involve many contributors. After a given time, the intake of new contributors does not deeply affect the number of contributors and the maximal values of Gini indices tend to become stable. However, if visualisations with limited time ranges are used, as suggested in the previous paragraph, the number of contributors taken into account may dramatically change and a direct study of their evolution could not be achieved.

The generalisability of our results is also subject to caution. One of the three studied projects (namely Wine) has a behaviour different from the two others. The number of studied projects is clearly not enough to generalise the results to the other OSS. In the same vein, even if a study of each of the projects belonging to GNOME reveals a evidence for the Pareto principle in this ecosystem, we cannot generalise our observations to other OSS ecosystems, such as KDE or Apache.

\section*{6.8 Conclusion}

In this article, we studied and compared the evolution of OSS project activity. Following the GQM paradigm, our main research goal was to understand how activity is distributed in OSS projects over time. We considered three categories of activity: committing data and code, sending mail and changing bug reports. We extracted and analysed such activity based on three different types of data sources: version repositories, mailing lists, and bug tracking data. We carried out an empirical analysis over three different long-lived OSS projects for which this data was available: Brasero, Evince, and Wine, as well as over the GNOME OSS ecosystem.

For all three studied projects and for all considered activity categories, as well as for the commit activity in GNOME, we found evidence for the Pareto principle. Our results
6.8. CONCLUSION

 seem to show a power law behaviour that must be statistically confirmed. The activity
distributions showed a strong inequality in the activity of different contributors involved
in an OSS project: there is a small group of very active members, and a much bigger
group of largely inactive members. For two of the three studied projects, the core group
of most active members takes part in more than one activity category. In Wine this was
much less the case.

In order to gain understanding in how OSS projects evolve, we studied this inequality
of distribution over time. To do so we relied on statistical techniques borrowed from
econometrics. We applied three economic aggregation measures (the Hoover, Gini, and
Theil index). The evolution of activity distribution appeared to follow two kinds of
behaviour. The first one is typical of a totally new project: at the beginning, the activity is
more or less equally distributed, but quickly we observe a tendency towards a more unequal
distribution where the activities become more concentrated in a small core team. In the
second type of observed behaviour, the activity distribution is already imbalanced since
the beginning of the project, and this imbalance continues to become more pronounced
over time.

Studying who are the most active persons involved in each type of activity, we discov-
ered that these persons are often very active in different activity categories. For Brasero
and Evince, the two projects in which we observed this behaviour, the project is led by a
small group of very active members wearing several hats at the same time. We have not
yet been able to identify the cause of overlaps between activities, because our definitions
for measuring activity need to be refined further.

While Brasero and Evince show a similar evolution of activity distribution and simi-
lar overlaps between most active persons’ activity categories, the Wine software project
appears to have a different behaviour. It has a significantly bigger community, there is
significantly less overlap between activity categories, and we observed a higher inequality
in the activity distribution. We can only speculate as to the causes of this.
On the specialisation of GNOME workload

This chapter presents a case study that focuses on the actual contributors to the GNOME ecosystem. We studied the workload of persons involved in the source code repositories of its projects with the aim to gain a better insight in the way the contributors specialise themselves. To this aim, we defined a new series of workload and involvement metrics, as well as a novel approach for reporting the results of comparing multiple distributions through the use of $T$-graphs. We used these techniques to statistically study how workload and involvement of GNOME contributors varies across projects and across activity types, and we explored to which extent projects and contributors specialise in particular activity types.

We observed that, next to coding, the activities of localisation, development documentation and building are prevalent throughout the ecosystem. We also observed notable differences between frequent and occasional contributors in terms of the activity types they are involved in and the number of projects they contribute to. Occasional contributors and contributors that are involved in many different projects tend to be more involved in the localisation activity, while frequent contributors tend to be more involved in the coding activity in a limited number of projects.

The contents of this chapter is mainly based on a previous publication in the Empirical Software Engineering Journal [172]. The statistical aspects addressed in this chapter, in particular those related to the comparison of multiple distributions, are mainly contributed by two co-authors of this publication, namely Bogdan Vasilescu and Alexander Serebrenik.


7.1 Introduction

In Chapter 6, we present an empirical study of the GNOME community and three open source projects. We distinguished activities in the source code repositories, the mailing lists and the bug trackers of the considered systems. In this chapter, we study more in detail the activity related to the source code repositories only, going beyond existing research in software repository mining and focusing on the community of GNOME contributors. In particular, we wish to get insight in the variation of workload and activity types across the contributors to the different projects that make up the ecosystem.

As seen in Chapter 6, all contributors of an ecosystem need to communicate, interact and collaborate in order to adapt and maintain the ecosystem and its constituent projects. However, some of these contributors are considerably more active than others, some contribute to multiple projects, and many are involved in different types of activities.

The social interactions between open source contributors, as well as their degree of project participation have been reported repeatedly to influence software quality and complexity [14, 41]. Such information needs to be carefully and empirically analysed in order to get a better understanding of how open source contributors interact as part of a large ecosystem built up from multiple interrelated projects.

Another important aspect that is largely unexplored in empirical analysis of software repositories is how contributors specialise themselves in a restricted number of activity types. As explained in Chapter 6, one can distinguish different activity types such as coding, development documentation, building, testing, and so on.

Both German [55] and the GNOME developers themselves recognised the importance of non-coding activities for GNOME, as well as contributors specialising themselves in these activities:

“GNOME Community Celebrates 10 Years of Software Freedom, Innovation and Industry Adoption: Since 1997, the GNOME project has grown from a handful of developers to a contributor base of coders, documenters, translators, interface designers, accessibility specialists, artists and testers numbering in the thousands.” [177]

“Just on this note, let me state that I in no way consider translators as second-class citizens; nor documenters, UI dudes, general organisers, or anyone whatsoever just because they do not code.” [160]
The aim of this case study is to explore the variation in workload of projects and contributors of GNOME, taking into account the activity types they are involved in.

This chapter is structured as follows. Section 7.2 presents our two research goals and explains the research methodology followed. We introduce a novel set of metrics to study the variation of workload and involvement, and we present $\mathcal{T}$-graphs as a novel approach to report the results of comparing multiple distributions. Sections 7.3 and 7.4 report on the statistical evaluation carried out for each research goal, and discuss the results. Section 7.5 presents the threats to validity, Section 7.6 reviews related work, and Section 7.7 concludes.

### 7.2 Methodology

#### 7.2.1 Research Goals

As mentioned in the introduction of this dissertation, we believe that studying the contributors to a software ecosystem (its ecosystem community) is equally important as studying the contributions to the ecosystem themselves. Therefore, we focus on participation of individual contributors, and study variations of the amount of participation across projects of the ecosystem and across contributors of the ecosystem community.

For this reason, we explore the following two research questions:

1. How does workload vary across projects of GNOME?
2. How does workload vary across contributors to GNOME?

We decided to use the term *workload* as an objective measure of the amount of participation. Its formal definition will be given in Section 7.2.4. As explained in the introduction, and as observed in Chapter 6 as well as in a previous study [113], the workload of projects or contributors may vary a lot depending on the *type of activity* that is being considered. Therefore we will take the type of activity into account while studying both research goals. In complement to what has been presented in Chapter 6, in this chapter we focus on activity types related to the changes committed in the source code repositories belonging to the GNOME projects.
Table 7.1: Variation of Git project characteristics across 1,316 GNOME projects. For each project, the number of commits, committers, and authors was computed for the entire considered project history. The number of files was computed for the last considered commit only.

Section 7.3 will study the first research goal, and Section 7.4 will study the second research goal. It is in these sections that we will formulate the research questions and how they contribute to each goal.

The research goals use the notion of contributor belonging to the ecosystem community. As explained in Section 3.2.2, Git repositories distinguish between committers and authors. For the reason evoked in Section 3.2.2, in the rest of this chapter, we will use the term author instead of contributor, to reflect the fact that we restrict our case study to only those persons that contribute to the Git project repositories of GNOME.

To extract relevant data from these Git repositories, we used CVSAnaly2, a specialised tool presented in Section 3.2.2, to produce databases containing all the information related to GNOME projects and contained in their source code repositories.

Table 7.1 shows how some project characteristics vary across GNOME projects. For each project we have retrieved the number of committers, number of commits, number of authors and number of files (that latter values are computed only for the latest commit retrieved for each project). Based on these values we computed the median, minimum, maximum, lower quartile (Q1), upper quartile (Q3) and mean values. The boxplots in Figure 7.1 visualise the distribution of these results.

### 7.2.2 Identity merging

As explained in Chapter 4, an identity merging tool must be applied on the authors accounts belonging to the source code repositories of GNOME projects in order to gain a better model of the stakeholders actually involved as authors.
Figure 7.1: Boxplots showing the variation of Git characteristics from Table 7.1 (log $y$-axis). Red triangles show the mean value.
In this study, we achieved an identity merging based on an adapted version of the *improved* algorithm described in Section 4.4.5. The adaptations consist of the relying on the data extracted from the Git code repositories only, as well as manual pre-processing and post-processing that aim to improve the quality of the obtained merge model. The measure has also been improved by using different algorithms, including the Levenshtein distance that was already used in the original improved algorithm. The different steps of the complete merging process are schematised in Figure 7.2.

**Figure 7.2: Identity matching steps.**

First, GNOME-specific naming artefacts, such as the timestamp prefixes in different formats, are identified by manual inspection (step 0.1), then the author names are automatically preprocessed to remove these prefixes (step 0.2).

Next, a list of candidate matches is computed for each name (step 1.1) using a number of similarity measures\(^1\) based on pattern matching (e.g., the Levenshtein distance [98, 122]), phonetic encoding (e.g., soundex [87]), or a combination of both (e.g., editex [187]). Christen [33] provides an overview of such similarity measures. Although numerous name similarity measures exist and have available implementations, e.g., as part of Fébrl [34], computing such measures is often computationally expensive. Moreover, there is no single

---

\(^1\)The implementations of the similarity measures are part of Fébrl – a parallel open source data linkage system [34].
best technique available. Therefore, we applied the Bertillonage approach proposed by Davies et al. [40] to reduce the search space using fast techniques, followed by more expensive computations on this reduced data set. Applying this to identity matching, we started by using a subset of the available similarity measures and only then performed a manual post-processing.

From the set of available similarity measures of Christen [33] we required a limited subset (to ensure fast computation), well-balanced in terms of complementary types of similarity measures (pattern matching, phonetic encoding, combinations of both, or more advanced measures), and containing techniques that have been shown to perform well in practice. In this sense we selected the Levenshtein distance, Damerau-Levenshtein distance, and bag distance pattern matching techniques, the soundex and phonex phonetic encoding techniques, the editex combined measure, and the Jaro and Winkler data linkage algorithms. All of these algorithms are presented in detail by Christen [33], who also shows experimentally that they perform well on real name data sets.

A name is considered a candidate match (step 1.2) when at least one of the selected similarity measures exceeds a certain threshold. For a given similarity measure and a given name, the higher the threshold, the fewer the candidate matches and, conversely, the lower the threshold, the more the candidate matches for that name. We observed that a threshold value of 0.8 offered a good tradeoff between the number of candidate matches and the number of false positives. If needed, the threshold can be changed, since it only impacts the amount of manual post-processing required.

Similarity measures are sensitive to the ordering of name parts (e.g., ‘Attila Hammer’ and ‘Hammer Attila’) and to the presence of middle names or initials (e.g., ‘Lars R. Clausen’ and ‘Lars Clausen’). They also fail to recognise as candidate matches login names corresponding to the same identity, even when logins are formatted according to commonly-adopted naming conventions, such as the first letter of the first name followed by the last name. Step (1.3) extends the list of candidate matches to incorporate these cases automatically.

Similarity measures are not necessarily transitive. In step (2.1), in order to have a complete list of candidates, we represent the candidate match relation as a graph in which names are nodes, and there is an edge between two nodes if one of them is a candidate match for the other. The sets of aliases used by the same authors is therefore the set of connected components of the graph.

In the manual post-processing step (3.1), remaining logins that do not adhere to commonly-adopted naming conventions (e.g., ‘mrhappypants’) are matched to existing
names by searching on the internet for email addresses used in common both by the nicknames (logins) and the existing names. All the matches are independently checked by different persons that are not directly involved in GNOME projects but having experience in open source software communities, without being aware of which matches were suggested by the algorithm or by the manual post-processing.

Applying the above identity matching approach to the data about the GNOME contributors allowed us to quantify the scale of the problem: without name matching, we found 6982 different author names across all considered GNOME projects. After name matching, only 5,155 unique identities remained (i.e., 73.83%). When counting the number of different names associated to these unique identities, we found a median value of 1, a mean value of 1.355, and a maximum value of 168. In fact, in 4,344 cases (i.e., 84.26%) the unique identities correspond to a single name. In 555 cases (i.e., 10.77%) the identities correspond to two different names. The remaining 4.97% unique identities correspond to persons that have used 3 or more different names to identify themselves. The maximum number of aliases (168) corresponded to an author that used commit messages instead of his name.

In the remainder of this chapter, whenever we use the term *author*, we refer to the unique identities obtained as a result of the identity matching process.

### 7.2.3 Activity types detection

Chapter 5 explained that the type of development activity carried out by authors in a project can be estimated on the basis of the types of files that are touched during each commit in the project’s version control repository. We will consider the 14 activity types defined and discussed in Chapter 5. However, in order not to distort our data, we do include unknown files in the computation of all our metrics.

Appendix B presents all the matching rules used in this chapter for achieving the activity type detection.

### 7.2.4 Metrics

Having defined the research goals, we now present a novel set of metrics that we have created to be able to answer the research questions for each research goal in Sections 7.3 and 7.4. These metrics are somehow restricted by the type of data that we can extract
from the different GNOME project repositories in reasonable time. We decided to focus on file-level metrics as the most suitable level of granularity for our case study. While analysing data below file level would allow us to be more precise, it turns out to be too time-consuming and resource-consuming. In addition, the file contents is only useful for text-based files such as code files, while a more in-depth analysis of code files would necessitate the use of different parsers (one for each language used).

Ignoring files by studying commits would be too coarse grained, as it does not allow us to approximate the workload of individual authors at a sufficient level of detail. In particular, it does not allow us to identify the different activity types carried out by authors (see Section 7.2.3), while this is a prerequisite for addressing the second research goal.

Figure 7.3: Workload metrics. The following naming convention is adopted for the metric acronyms: A = Author; P = Project; T = activity Type; W = Workload; R = Relative; S = Specialisation. Relative metrics are defined as a fraction and represent a percentage (i.e., a value between 0 and 1). \( Gini_T \) denotes the application of the \( Gini \) inequality index over all activity types, as described in Section 6.3.3. Similarly, \( Sum_T \), \( Sum_A \) and \( Sum_P \) aggregate values through summation.

Let \( P \) be the set of all GNOME projects, \( A \) the set of all unique GNOME authors (i.e. the GNOME contributors after matching different logins to the same identity), \( T \) the set of all considered activity types. Each file belonging to some commit in the version control repository of a project \( p \in P \) can be directly linked to an author \( a \in A \) that touched this file, and the type \( t \in T \) of the activity corresponding to this file is computed as explained in Chapter 5.
The basic metric we compute using data extracted from the Git logs is the **Author-Project-Type Workload** $APTW$:

$$APTW(p, a, t) = \text{number of touches to files of activity type } t$$

by author $a$ for project $p$ over its entire history. (7.1)

If the same file is touched in different commits, it will be counted multiple times. Based on this metric, we can also derive the **Author-Project-Type Involvement** $APTI$ that determines for project $p$ if an author $a$ has been involved in at least one (i.e., has touched at least one file of) activity type $t$:

$$APTI(p, a, t) = \begin{cases} 1, & \text{if } APTW(p, a, t) > 0; \\ 0, & \text{otherwise.} \end{cases}$$ (7.2)

Using these two basic metrics, we can derive higher-level aggregate metrics. Figure 7.3 presents the workload metrics that are derived from $APTW$, while Figure 7.4 presents the involvement metrics that are derived from $APTI$. In both figures we distinguish between project-level metrics (on the left) and author-level metrics (on the right).

Figure 7.4: Involvement metrics. The same naming convention is followed as in Figure 7.3, except that we now use $I$ for *involvement* and $N$ for *number of*.

The way these metrics are computed is similar. We therefore only present the project-level metrics definitions in Tables 7.2 and 7.3.
The main distinction between workload metrics and involvement metrics is that the latter rely on counting. For example, \( NAP(p) \) counts how many authors are involved in project \( p \). If the same author is involved in different activity types for this project, she needs to be counted only once. This explains why we first compute the maximum over all types, and then compute the sum over all authors.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( PTW(p,t) )</td>
<td>(absolute) project-type workload</td>
<td>[ \sum_{a_j \in A} APTW(p,a_j,t) ]</td>
</tr>
<tr>
<td>( PW(p) )</td>
<td>project workload over all authors and activity types</td>
<td>[ \sum_{t_k \in T} PTW(p,t_k) ]</td>
</tr>
<tr>
<td>( TW(t) )</td>
<td>type workload over all authors and projects</td>
<td>[ \sum_{p_i \in P} PTW(p_i,t) ]</td>
</tr>
<tr>
<td>( RPTW(p,t) )</td>
<td>workload in project ( p ) for activity type ( t ), relative to the total project workload</td>
<td>[ \frac{PTW(p,t)}{PW(p)} ]</td>
</tr>
<tr>
<td>( RTPW(p,t) )</td>
<td>workload in project ( p ) for activity type ( t ), relative to the total type workload</td>
<td>[ \frac{PTW(p,t)}{TW(t)} ]</td>
</tr>
<tr>
<td>( PWS(p) )</td>
<td>specialisation (imbalance) of workload across activity types for project ( p ), over all authors contributing to ( p )</td>
<td>[ Gini_{t_k \in T} (PTW(p,t_k)) ]</td>
</tr>
<tr>
<td>( RPWS(p) )</td>
<td>specialisation (imbalance) of relative workload across activity types for project ( p ), over all authors contributing to ( p )</td>
<td>[ Gini_{t_k \in T} (RPTW(p,t_k)) ]</td>
</tr>
</tbody>
</table>

Table 7.2: Definitions of \( APTW \)-based project-level workload metrics. (The author-level workload metrics are defined similarly.)

Tables 7.2 and 7.3 and Figures 7.3 and 7.4 also refer to specialisation metrics that need some further explanation. To quantify the degree of specialisation of authors (towards a particular activity type), as well as the degree of project specialisation (towards a particular activity type), we rely on the econometric Gini inequality index that has been presented in Section 6.3.3. We have used it to define the project specialisation metrics \( PWS, RPWS, PIS, \) and \( RPIS \), as well as the author specialisation metrics \( AWS, RAWS, AIS \) and \( RAIS \) by aggregating over all activity types in \( T \).

Finally, we have chosen to focus on the specialisation of authors and projects towards a particular activity type. Alternatively, one could have studied specialisation of authors towards a particular project \( Gini_{a \in A} \left( \sum_{t \in T} APTW(p,a,t) \right) \) (similar to the project
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTI(p,t)</td>
<td>(absolute) project-type involvement</td>
<td>$\sum_{a_j \in A} APTI(p,a_j,t)$</td>
</tr>
<tr>
<td>NTP(p)</td>
<td>number of types for project p</td>
<td>$\sum_{t_k \in T} \max_{a_j \in A}(APTI(p,a_j,t_k))$</td>
</tr>
<tr>
<td>NAP(p)</td>
<td>number of authors for project p</td>
<td>$\sum_{a_j \in A} \max_{t_k \in T}(APTI(p,a_j,t_k))$</td>
</tr>
<tr>
<td>RPTI(p,t)</td>
<td>author involvement in project p for activity type t, relative to the total number of authors involved in the project</td>
<td>$\frac{PTI(p,t)}{NAP(p)}$</td>
</tr>
<tr>
<td>PIS(p)</td>
<td>specialisation (imbalance) of involvement across activity types for project p, over all authors contributing to p</td>
<td>$Gini_{t_k \in T}(PTI(p,t_k))$</td>
</tr>
<tr>
<td>RPIS(p)</td>
<td>specialisation (imbalance) of relative involvement across activity types for project p, over all authors in p</td>
<td>$Gini_{t_k \in T}(RPTI(p,t_k))$</td>
</tr>
</tbody>
</table>

Table 7.3: Definitions of $APTI$-based project-level involvement metrics. (The author-level involvement metrics are defined similarly.)

Work Concentration measure of Tsay et al. [165]) or specialisation of projects towards a particular author $Gini_{p \in P} \left( \sum_{t \in T} APTW(p,a,t) \right)$.

### 7.2.5 Data analysis

In order to facilitate replication of our case study, we have created a webpage and a replication package\(^2\) containing the data, tooling, and detailed results of the statistical analysis performed. In this section we briefly introduce the techniques we have used to perform statistical analysis. We relied on the R project for statistical computing, including packages such as **ineq** to calculate the Gini index [185], **Matching** to perform the bootstrapped Kolmogorov-Smirnov test [147], **agricolae** to determine the Kendall correlation coefficient [42], and **nparcomp** to compute relative contrast effects when comparing two distributions [89].

---

\(^2\)The dataset can be found here: [www.win.tue.nl/mdse/gnome](http://www.win.tue.nl/mdse/gnome)
7.2. METHODOLOGY

Correlation When measuring statistical correlation between two groups of data we have a choice between linear or rank correlation coefficients. Linear coefficients (e.g., [130]) are sensitive only to a linear relation between two variables. Rank coefficients [84, 156] are more robust to nonlinear relations since they only measure the extent to which an increase in one variable (not necessarily linear) corresponds to an increase in the other variable. Since we do not make assumptions about the shape of each relation, we use a rank coefficient and we opt for Kendall’s $\tau$ since Spearman’s $\rho$ is known to be difficult to interpret [127]. We account for ties as described by Press et al. [134]. Whenever we measure Kendall correlation between two metrics, the null hypothesis $H_0$ is that there is no relation between the two metrics, and the alternative hypothesis $H_a$ is that there is a relation between the two metrics. We report Kendall’s $\tau$ and the corresponding $p$-value.

Linear regression When a linear relation between the dependent variable and one or more independent variables can be suspected, we also perform linear regression, i.e. based on the data we estimate parameters of the linear function of the independent variables to obtain as close values as possible to the values of the dependent variable. To check the adequateness of the fitted model we analyse the residual plot: the points in the residual plot should appear randomly dispersed around the horizontal axis. Moreover, we report the $p$-values for the significance of regression with the $F$-statistic, as well as $p$-values for the coefficients and the intercept. Finally, we report the adjusted coefficient of determination $\bar{R}^2$ [164, pp. 164,175–178] that takes into account the number of parameters used by the regression model.

Distribution fitting In order to understand how data values are distributed, we try to fit a theoretical distribution to it. Specifically, as many distributions in software follow a power law $x^{-\alpha}$ [104] or are log-normal [12, 101], in this chapter we only attempt to fit these types of distributions. To evaluate the goodness-of-fit of a log-normal distribution we use the Kolmogorov-Smirnov test. The original test cannot calculate correct $p$-values in presence of ties. In those cases we use the bootstrapped version of the Kolmogorov-Smirnov test [147] instead. In this test we use the two-sided alternative hypothesis and the default number of bootstraps to be performed (1000). Considering a power law distribution, it often applies only for values greater than some minimum value, so we need to estimate this value in addition to $\alpha$ that determines the form of the distribution. Using the methodology proposed by Clauset, Shalizi and Newman [35] we estimate the aforementioned parameters and calculate the goodness-of-fit between the data and the power law. If the resulting $p$-value is lower than the threshold of 0.1 proposed by [35], we reject the hypothesis that the distribution follows a power law. If the $p$-value is higher than 0.1, it is possible that other distributions can be fitted as well. Therefore, we have to
compare the likelihood of the data under two competing distributions. Depending on the families these distributions belong to, we either exploit the closeness test of Vuong [176] or a slightly modified likelihood ratio test [35].

Excluding zeros As part of our research goals we study the influence of the activity type (e.g., coding, localisation) on workload variations across projects and across project contributors. To this end we distinguish between, and compute metrics per, different activity types. Whenever we compute a metric that takes the activity type into account, we consistently exclude zero values in order to do not take into account the contributors who have never been involved in the considered activity types in the workload distribution. For each activity type, we only focus on the projects (contributors) that contain (participate in) activities of that type, cf. discussion of active committers of [139]. The only exception is when we compare specialisation of projects and contributors in few activity types (i.e., Figures 7.8 and 7.14), computed using the Gini index. In these cases, since not all projects contain, and not all contributors participate in, activities of all types, we do not exclude zero values (e.g., for a project we do not ignore the activity types not present in that project), since this would lead to incomparable Gini index values.

7.2.6 $\tilde{T}$ procedure and $\tilde{T}$-graph

When studying the specialisation of projects and authors towards different activity types, we need to assess whether the distributions of a given metric are different for the different activity types. Traditionally, comparison of multiple groups follows a two-step approach: first, a global null hypothesis is tested, and then multiple comparisons are used to test sub-hypotheses pertaining to each pair of groups. The first step is commonly carried out by means of ANOVA or its non-parametric counterpart, the Kruskal-Wallis one-way analysis of variance by ranks [75]. The second step uses the $t$-test or the rank-based Wilcoxon-Mann-Whitney test [179], with Bonferroni correction [48, 150]. Unfortunately, the global test null hypothesis may be rejected while none of the sub-hypotheses are rejected, or vice versa [54]. Moreover, simulation studies suggest that the Wilcoxon-Mann-Whitney test is not robust to unequal population variances, especially in the unequal sample size case [186]. Therefore, one-step approaches are preferred: these should produce confidence intervals which always lead to the same test decisions as the multiple comparisons.

Moreover, since we have identified 13 different activity types\(^3\), we had to conduct \(\frac{13\cdot12}{2} = 78\) comparisons and report 78 results. For the sake of brevity we summarise the

\(^3\)As explained in Section 7.2.3 we do not include the unknown activity type.
test results as a directed acyclic graph. Nodes of the graph correspond to activity types, edges to results of pairwise comparisons. Because plotting a graph with 13 nodes and in the worst case 78 edges would result in visual clutter, we would like to omit direct edges between $A$ and $B$ if there is a path from $A$ to $B$ passing through at least one other node $C$. Hence, we need an approach that respects transitivity. Unfortunately, this is not necessarily the case for traditional pairwise or multiple comparison approaches: e.g. Brown and Hettmansperger [19] show that no transitive reduction is possible for the traditional pairwise Wilcoxon-Mann-Whitney tests. Transitivity is, however, respected by the recently proposed multiple contrast test procedure $\tilde{T}$ [90]. Moreover, $\tilde{T}$ is robust against unequal population variances.

The $\tilde{T}$ procedure takes as input a type of contrast and the threshold for the family-wise error rate, i.e., the probability of falsely rejecting one or more null sub-hypotheses [94] (we use the traditional threshold of 5%). The $\tilde{T}$ procedure returns an estimator for the difference of each pair of the distributions being compared, the corresponding 95% confidence interval, test statistics and the corresponding $p$-values.

Contrasts, represented as the contrast matrix, express which sub-hypotheses should be tested. Formally, matrix $C$ is called a contrast matrix if $C \cdot 1 = 0$, where $1$ is the column vector of appropriate length consisting solely of ones and $0$ is the row vector consisting solely of zeroes, i.e. the sum of all rows in $C$ is 0 [21]. To illustrate the notion of a contrast matrix consider the following matrices:

$$C_D = \begin{pmatrix} -1 & 1 & 0 & \ldots & 0 \\ -1 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & 0 & 0 & \ldots & 1 \end{pmatrix} \quad C_T = \begin{pmatrix} -1 & 1 & 0 & \ldots & 0 & 0 \\ -1 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & 0 & 0 & \ldots & 0 & 1 \\ 0 & -1 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & -1 & 0 & \ldots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & -1 & 1 \end{pmatrix}$$

Matrix $C_D$ expresses comparisons of multiple alternative hypotheses (treatments) with a specific one (control group) and is known as “many-to-one” or Dunnett-type contrast [49]. Matrix $C_T$ expresses all pairwise comparisons (up to symmetry) and is known as “all pairs” or Tukey-type contrast [166]. Since our goal is to compare all groups pairwise, we consider only Tukey-type contrasts.

Next we introduce $\tilde{T}$-graphs, a new and more intuitive visualisation that we propose for reporting the results of the $\tilde{T}$ procedure:
• First, for each pair of groups we analyse the 95% confidence interval to test whether the corresponding null sub-hypothesis can be rejected. If the lower boundary of the interval is greater than zero for groups \( A \) and \( B \), then we claim that the metric value is higher in \( A \) than in \( B \). Similarly, if the upper boundary of the interval is less than zero for groups \( A \) and \( B \), then we claim that the metric value is lower in \( A \) than in \( B \). Finally, if the lower boundary of the interval is less than zero and the upper boundary is greater than zero, we conclude that the data does not provide enough evidence to reject the null hypothesis.

• Second, based on the results of the comparisons we construct the graph with nodes being groups and containing edges \((A, B)\) if the metric value is higher in \( A \) than in \( B \). After removal of transitive edges \([2]\), we obtain a directed acyclic graph that we call a \( \tilde{\mathbf{T}} \)-graph.

A visual comparison of multiple distributions using \( \tilde{\mathbf{T}} \)-graphs enables us to focus on “interesting” groups, e.g., activity types located “high” in the graph, i.e. those activity types with metric values higher than most of the remaining activity types, or “low” in the graph, i.e. those activity types with metric values lower than many remaining activity types.

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Developers</th>
<th>Pair</th>
<th>Lower</th>
<th>Upper</th>
<th>( p )-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2 2 2 3 3 3 3 3 4 4 4 4 4 4 4 4 5 5</td>
<td>B–A</td>
<td>-0.560</td>
<td>-0.444</td>
<td>0.000</td>
</tr>
<tr>
<td>B</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2</td>
<td>C–A</td>
<td>-0.503</td>
<td>-0.313</td>
<td>7.536e-10</td>
</tr>
<tr>
<td>C</td>
<td>1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 3 3</td>
<td>D–A</td>
<td>-0.320</td>
<td>-0.027</td>
<td>1.997e-02</td>
</tr>
<tr>
<td>D</td>
<td>1 1 1 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4</td>
<td>C–B</td>
<td>0.014</td>
<td>0.242</td>
<td>9.742e-02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D–B</td>
<td>0.237</td>
<td>0.470</td>
<td>1.200e-06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D–C</td>
<td>0.090</td>
<td>0.404</td>
<td>2.432e-03</td>
</tr>
</tbody>
</table>

Table 7.4: Illustration of \( \tilde{\mathbf{T}} \) procedure and \( \tilde{\mathbf{T}} \)-graph based on artificial data. Left: Commit activity per type for 20 developers (columns). Middle: Results of the \( \tilde{\mathbf{T}} \) procedure. The \( p \)-value reported as zero is too small to be calculated exactly. Right: The resulting \( \tilde{\mathbf{T}} \)-graph.

To illustrate the \( \tilde{\mathbf{T}} \) procedure and a \( \tilde{\mathbf{T}} \)-graph consider the following artificial example inspired by and extending the drug\(_1\) data of Akritas et al. [3]. Figure 7.4 (left) shows the commit activity per activity type (\( A, B, C \) or \( D \)) for a group of twenty developers: e.g. developer \( \#1 \) has performed two commits for activity \( A \), one commit for activity \( B \), one commit for activity \( C \) and one commit for activity \( D \). Using the \( \mathbf{T} \) procedure and a \( \mathbf{T} \)-graph we would like to clarify the relationship between the four activity types. We start by invoking the \( \mathbf{T} \) procedure for the Tukey-type contrast and 95% confidence level. Results of the \( \mathbf{T} \) procedure are summarised in Figure 7.4 (middle). For five out of six comparisons
the \( \tilde{T} \) procedure reports \( p < 0.05 \) or, equivalently, the corresponding 95% confidence interval does not contain zero. Since the lower boundary of the confidence interval for D–B and D–C is greater than zero, the corresponding graph should contain edges from D to B and from D to C. Similarly, since the upper boundary of the confidence interval for B–A, C–A and D–A is smaller than zero, the corresponding graph should contain edges from A to B, A to C and A to D. After removal of transitive edges we obtain the \( \tilde{T} \)-graph with three edges shown in Figure 7.4 (right).

A special case of comparison of multiple distributions is the comparison of two distributions. We need to test whether one of two samples of independent observations tends to have larger values than the other. Traditionally, distributions of software metrics have been compared using the Wilcoxon-Mann-Whitney two-sample rank-sum test [6, 85]. However, Wilcoxon-Mann-Whitney is not robust against differences in variance [186, 20]. The \( \tilde{T} \) procedure as described above cannot be applied to comparison of two distributions [90]. We therefore prefer the two-distributions equivalent of the \( \tilde{T} \) procedure, i.e., we perform two sample tests for the nonparametric Behrens-Fisher problem [20], and compute confidence intervals for the relative effect of the two samples. If the relative effect \( p(a,b) > 0.5 \) then \( b \) tends to be larger than \( a \). Moreover, since software metrics are frequently being compared using the Wilcoxon-Mann-Whitney two-sample rank-sum test [6, 85], we also report the results of this test.

7.3 Goal 1: How does workload vary across projects?

Our first research goal consists in understanding how workload varies across projects belonging to the same ecosystem. In order to address this goal we study cross-project variation of measurable project-level properties (e.g., project workload \( PW \), number of authors involved in a project \( NAP \), number of activity types per project \( NTP \)) by answering the following research questions:

1. How does project workload vary across the ecosystem?
2. Which types of projects are more active?
3. How specialised are projects towards different activity types?
4. What are the characteristics of specialised projects?
7.3.1 How does project workload vary across the ecosystem?

We start by studying the variation of the project workload $PW(p)$ across the ecosystem, for all $p \in P$. The distribution is left-skewed and the maximal value is more than an order of magnitude larger than the mean: two features typical for heavy-tailed distributions [162]. We first hypothesise that the project workload follows a power law. This hypothesis can be rejected since the $p$-value of the goodness-of-fit test equals 0.0496, which is lower than the threshold of 0.1 [35]. Next, we consider the log-normal distribution. Since the data contains ties we opt for the bootstrapped Kolmogorov-Smirnov test [147]. The corresponding $p$-value equals 0.533, and, hence, the hypothesis that the project workload follows the log-normal distribution cannot be rejected. A histogram of log $PW(p)$ is presented in Figure 7.5.

![Histogram of log(PW)](image)

Figure 7.5: The workload $PW(p)$ is distributed log-normally.

Project workload is distributed log-normally across the software ecosystem.

Figure 7.5 also reveals exceptional projects. At the lower end of the scale we distinguish archived projects, and projects with very little activity. Further inspection of the commit logs and GNOME mailing list archives revealed that since some of the latter modules have not seen any recent activity or are closed in the issue tracker for new bug entries, they are likely to be archived soon as well. This was for example the case for gnome-audio, that
7.3. **GOAL 1: HOW DOES WORKLOAD VARY ACROSS PROJECTS?**

had very little activity until October 2011 (the latest date considered in our case study) and is indeed listed as archived in October 2012. Other projects with small workload either have incomplete repositories, potentially as a result of migration from CVS to Git (e.g., *O3web*), or are auxiliary (e.g., *perl-Clutter* which, although stand-alone, represents only a set of Perl bindings for Clutter 1.x). At the higher end of the scale we distinguish very active projects such as *GIMP*, the GNU image manipulation program, or *Evolution*, the email, contacts and scheduling manager.

### 7.3.2 Which projects are more active?

**Are projects containing more activity types more active?**

To study this first question, we compare the number of activity types per project $NTP(p)$ and the project workload $PW(p)$. With a Kendall correlation test we observe a strong correlation ($\tau = 0.6$), and reject $H_0$ ($p$-value $< 2.2 \times 10^{-16}$). Closer inspection of the scatter plot in Figure 7.6 suggests a linear relation between $NTP(p)$ and $log\ PW(p)$. Using linear regression we obtain the model $log\ PW(p) = 0.64562 \times NTP(p) + 1.57412$ ($R^2 = 0.6129$). The fitted linear model is adequate: $F$-statistic equals 2109 on 1 and 1314 degrees of freedom with the corresponding $p$-value $< 2.2 \times 10^{-16}$, $p$-values for the coefficient and the intercept do not exceed $2.2 \times 10^{-16}$. The points in the residual plot appear randomly dispersed around the horizontal axis. We conclude that the project activity increases exponentially (due to the use of log $PW$ in the formula) as projects include more activity types: increasing the number of activity types by one increases the effort almost twice ($e^{0.64562} \approx 1.9$).

![Figure 7.6: Linear relation between $NTP(p)$ and log $PW(p)$](image-url)

(a) Observed linear relation between $NTP(p)$ and log $PW(p)$ (regression line drawn in red).

(b) Residuals plot.
The more activity types a project contains, the more active it is: increasing the number of activity types by one approximately doubles the project workload.

Figure 7.6 also reveals exceptional projects, being either very diverse (e.g., the Anjuta integrated development environment and the Banshee music player both contain activities of all 13 types considered), or very specialised (e.g., GTK tutorial is an archived project associated with the GTK toolkit for creating graphical user interfaces, and contains only documentation activities, while O3web is an archived project containing only build activities). The 18 projects containing activities of a single type are all categorised as archived.

Are projects with larger communities more active?

Does the number of authors $NAP(p)$ involved in project $p$ influence the total workload $PW(p)$? As a result of Kendall’s correlation test we observe a strong correlation between $NAP(p)$ and $PW(p)$ ($\tau = 0.64$) and reject $H_0$ (p-value $< 2.2 \times 10^{-16}$), suggesting that project workloads are higher as more authors are involved in the projects. We do not describe the relation between $NAP(p)$ and $PW(p)$ further since we could not obtain adequate linear regression models, for which the points in the residual plot would appear randomly dispersed around the horizontal axis.

The larger its community of contributors, the more active the project.

We observed some exceptional projects. For example, 218 projects (16.56%) are developed by a single author. Among them we find projects such as GSAPI and GSpeech (variations on the Java Speech API), which eventually became archived and were refined into Gnome Speech, which has a larger community of 15 authors. There are also non-archived projects developed by a single author. For example, Grits, a Virtual-Globe-like library that handles coordinates and the OpenGL viewport, is still actively maintained today by a single developer.

7.3.3 How specialised are projects towards different activity types?

Let us first explore how the ecosystem workload varies across the different activity types. Figure 7.7 displays this variation using the type workload $TW(t)$ aggregated over all projects. We observe a high inequality between the different activity types. code, devdoc,
110n, and build account for the highest shares of the ecosystem workload, representing together 78% of the total workload. All code activities by themselves account for more than 40% of the total workload.

Across the ecosystem, code, devdoc, l10n, and build activities account for the highest share of the workload. code is the predominant activity type.

Figure 7.7: Type workload: activity types code, devdoc, l10n, and build account for the highest workload share in the ecosystem.

Figure 7.8: Relative project workload specialisation RPWS(p): Most projects concentrate their workload in few activity types.
To what extent are projects specialised in few activity types?

The specialisation $RPWS(p)$ of a project $p$ can be interpreted as how a project $p$ specialises its relative workload $RPTW(p,t)$ towards few activity types $t$. It is computed by applying the Gini index to aggregate the $RPTW(p,t)$ values over all types $t \in T$. A high value of $RPWS(p)$ reflects a high inequality in the distribution of workload across the different activity types for project $p$. This suggests that most of the project’s workload is concentrated in few activity types, while the remaining activity types only account for a very small fraction of the workload. A low value of $RPWS(p)$ reflects a more equal distribution of the project’s workload across the different activity types.

Figure 7.8 displays the variation across projects of $RPWS(p)$. We observe that most projects are specialised in few activity types, since the median is high (0.77). Our observation is similar to the findings of Vasa et al. [171], according to which the typical overall range of Gini coefficients for multiple software metrics is between 0.45 and 0.75, thus values above 0.75 can be considered high. The highest values of $RPWS(p)$ have been observed for projects such as $O3web$ that focus on one activity only. For these projects $RPWS(p)$ reaches the highest theoretically possible value for Gini coefficient on a data set of 14 elements, i.e. $\frac{13}{14} \approx 0.93$.

The lowest value of $RPWS(p)$ is 0.459, i.e. it still belongs to the $[0.45, 0.75]$ range of Gini coefficients observed by Vasa et al. [171]. The lowest value of $RPWS(p)$ has been observed for $gnome-applets$, the project that distributes the activity in a most egalitarian way. $Gnome-applets$ is a collection of small unrelated applications for the GNOME desktop, including various monitors, weather report, trash bin and eyes following the mouse pointer around the screen. The second lowest $RPWS(p)$ value (0.555) was obtained for $gnome-utils$, another collection of small unrelated desktop applications.

Closer inspection of the $RPTW$ values for $gnome-applets$ and $gnome-utils$ reveals that both projects have a relatively high share of the build activity: 13% and 11%, respectively. This can be explained by the fact that one should be capable of compiling separately individual applications comprising $gnome-applets$ and $gnome-utils$, implying that each one of the application has its own makefile and related files.

Moreover, since $gnome-applets$ and $gnome-utils$ comprise desktop applications, a relatively high part of the effort is dedicated to $110n$ and $image$. All this leads to relatively egalitarian workload distribution, reflected in relatively low $RPWS$ values.

Most of a project’s workload is concentrated in few activity types.
Note that not each activity type is present in each project (e.g., not all projects contain db activities). However, since we are interested in comparing specialisation for different projects (i.e., comparing Gini index values, computed for $RPTW(p, t)$ data over all activity types $t \in T$), we consider for each project the set of all possible activity types. As explained in Section 7.2.5, we do not ignore the activity types $t$ for which $RPTW(p, t) = 0$ when computing $RPWS(p)$ since this would render Gini index values incomparable.

To what extent are projects specialised towards different activity types?

**Workload** The specialisation of a project $p$ towards a certain activity type $t$ can be expressed in terms of the relative project’s workload $RPTW(p, t)$, defined as the workload in project $p$ for activity type $t$ relative to the total workload in $p$. High $RPTW(p, t)$ values reflect that most of the workload of project $p$ is concentrated in $t$, i.e., $t$ is a predominant activity type in $p$ in terms of number of file touches. Similarly, low $RPTW(p, t)$ values reflect that activities of type $t$ are but auxiliary in $p$.

Figure 7.9 illustrates the variation across projects of $RPTW(p, t)$ for each activity type $t$. In each boxplot, we only consider the projects for which activity type $t$ is present (i.e., the workload $PTW(p, t) > 0$), since we are only interested in understanding how the workload varies for projects that contain activities of that type. The number of projects (out of the total 1,316 projects considered) for which $PTW(p, t) > 0$ is displayed below each activity type in the boxplot.

We observe two groups of activity types. On the one hand, code, build, devdoc, and l10n (the same four main activity types observed at ecosystem level, Figure 7.7) have the highest values, with code being the predominant one in the $\bar{T}$-graph. Since the median for $RPTW(p, code)$ is slightly less than 0.5, we can say that in most projects that contain coding activities, coding represents around 50% of the workload. There are 54 (4.1%) projects that do not contain any code activities at all. Further manual investigation of the source code repositories and mailing list archives revealed that such projects without code activities are often auxiliary, e.g., Gnome Backgrounds—a collection of desktop background images, or Gnome Cookbook—a cookbook used and developed by the GNOME community. On the other hand, lib, media, and db have the lowest $RPTW(p, t)$ values, all being leafs in the $\bar{T}$-graph.

At the level of individual projects, most of the workload is concentrated in code, followed by the devdoc, l10n, and build activity types.
respect to Tukey-type contrasts and 5% family-wise error rate shows differences between
the activity types in the T-graph on the right (cf. Section 7.2.5).

Figure 7.9: Boxplots for relative project workload per activity type \( t \). Per boxplot, zero
values are excluded. \texttt{code} is the predominant activity type at project level: in most
projects that contain coding, it represents around 50% of the workload. \texttt{devdoc}, \texttt{l10n},
and \texttt{build} each account for 10-20\% of the workload on average. The T-procedure with

\begin{verbatim}
\texttt{Figure 7.9: Boxplots for relative project workload per activity type \( t \). Per boxplot, zero
values are excluded. \texttt{code} is the predominant activity type at project level: in most
projects that contain coding, it represents around 50\% of the workload. \texttt{devdoc}, \texttt{l10n},
and \texttt{build} each account for 10-20\% of the workload on average. The T-procedure with
respect to Tukey-type contrasts and 5\% family-wise error rate shows differences between
the activity types in the T-graph on the right (cf. Section 7.2.5).}
\end{verbatim}
**Workforce**  Another way to express the specialisation of a project towards a certain activity type \( t \) is in terms of the relative project’s *workforce* \( RPTI(p, t) \), defined as the number of authors of \( p \) involved in activity type \( t \) relative to the total number of authors in \( p \). High \( RPTI(p, t) \) values reflect that most of the authors involved in \( p \) are contributing to activities of type \( t \). Low \( RPTI(p, t) \) values indicate that activities of types \( t \) that only attract a small fraction of the authors involved in \( p \).

The variation of \( RPTI(p, t) \) across projects per activity type \( t \) is illustrated in Figure 7.10. Similarly as before, we only consider the projects for which there is at least one author involved in activities of type \( t \), i.e., the involvement \( PTI(p, t) > 0 \) (their number is displayed below each activity type in Figure 7.10). On the one end of the spectrum we observe that 110n and devdoc (which encompasses updating the ChangeLog, a common practice of authors whenever they perform changes) attract most of the authors involved in projects (they are the dominant activity types in the \( \tilde{T} \)-graph), followed by build and only then code. On the other end of the scale we observe activity types such as lib, db, and media (the bottommost activity types in the \( \tilde{T} \)-graph), in which for most projects only a small fraction of the authors are involved.

| 110n and devdoc activities attract most of the authors involved in projects, followed by build and then code. |

We illustrate the variation of \( RPTI(p, t) \) across projects by taking a closer look at two projects, Gevice, GNOME’s Network Device Manager, and Evolution, GNOME’s contact manager, address manager and calendar. The \( RPTI \) values of Gevice are extremely high: \( RPTI(Gevice, t) = 1 \) for all \( t \) but lib, media, and test, meaning that for any other activity 100% of the Gevice authors are involved in it. This is not surprising since Gevice has only a single author. As opposed to Gevice, the \( RPTI \) values of Evolution are always lower than 1, i.e. there is no activity that would attract all 723 Evolution authors.

### 7.3.4 What are the characteristics of specialised projects?

In order to understand the characteristics of highly-specialised projects, we study the correlation between metrics representing the specialisation of projects, i.e. \( RPWS(p) \), \( PTW(p, t) \), \( RPTW(p, t) \), \( PTI(p, t) \) and \( RPTI(p, t) \), on the one hand, and general project characteristics, i.e. project workload \( PW(p) \), number of authors involved in a project \( NAP(p) \) and number of activity types per project \( NTP(p) \), on the other hand.
CHAPTER 7. ON THE SPECIALISATION OF GNOME WORKLOAD

Figure 7.10: Boxplots for relative project involvement per activity type $t$. Per boxplot, zero values are excluded. l10n attracts the highest fraction of the project authors. db activities are performed only by a small fraction of the project authors. The multiple contrast test procedure $T$ with respect to Tukey-type contrasts and 5% family-wise error rate shows differences between the activity types in the directed acyclic graph on the right (cf. Section 7.2.5).
7.3. GOAL 1: HOW DOES WORKLOAD VARY ACROSS PROJECTS?

Which project characteristics are observed when it is specialised towards few activity types?

In Section 7.3.3 we observed that, while some GNOME projects exhibit relatively low specialisation values (e.g. 0.459 for `gnome-applets`), the opposite is true for other projects (e.g. 0.93 for `O3web`). In this section we investigate which characteristics of a project influence its specialisation. We expect that projects with more workload (measured by $PW(p)$), as well as projects with larger communities (measured by $NAP(p)$) tend to be less specialised since more opportunities for diversity arise from higher workload and more authors. On the other hand, it is unclear whether more specialised projects consist of few activity types (which thus concentrate the workload), or consist of many activity types of which only few concentrate the workload. In order to answer the question we study Kendall correlation between $RPWS(p)$ and each one of the project-specific $PW(p)$, $NTP(p)$, and $NAP(p)$ metrics.

For $PW(p)$ we confidently reject $H_0$ at 0.01 significance level ($p$-value $= 1.26 \times 10^{-13}$). However, the correlation coefficient is very small and negative ($\tau = -0.13$), indicating a very weak relation between $RPWS(p)$ and $PW(p)$. For $NAP(p)$ we again confidently reject $H_0$ at 0.01 significance level ($p$-value $= 7.33 \times 10^{-37}$), and observe a small negative correlation ($\tau = -0.24$). We hence did not find conclusive evidence that projects with more activity or projects with larger communities tend to be less specialised.

Finally, we confidently reject $H_0$ at 0.01 significance level for $NTP(p)$ ($p$-value $= 6.85 \times 10^{-30}$) and observe a slightly higher negative correlation ($\tau = -0.39$). This suggests that the more activity types are present in a project, the lower the project’s specialisation towards those activity types, as measured by $RPWS(p)$. It follows that highly unequal distributions of workload across different activity types are due to few activity types being present in a project and thus the project’s workload being concentrated in those types, rather than many activity types being present in a project, with most of the project workload concentrated in one of these types.

Highly specialised projects comprise few activity types (as opposed to many activity types out of which only few would concentrate the workload). In contrast, we have not found enough evidence that projects with more workload or larger communities are less specialised towards few activity types.
To what extent does project community size relate to the workload (share) for a particular activity type?

In Section 7.3.2 we observed that projects with larger communities have higher workloads. We wish to understand how the size of a project community (measured by $NAP(p)$) relates to the workload $PTW(p,t)$ and the workload share $RPTW(p,t)$ of this project generated for a particular activity type $t$.

To answer the question we compute Kendal correlation between $NAP(p)$ and $PTW(p,t)$ on the one hand, and between $NAP(p)$ and $RPTW(p,t)$ on the other hand, for all activity types $t \in T$. For each activity type $t$, we only look at projects for which $PTW(p,t) > 0$, as discussed in Section 7.3.3. The results of the correlation tests are visually summarised in Figure 7.11a, page 131, for $PTW$ (drawn in black) and $RPTW$ (drawn in gray). The shape and fill of a point represent the $p$-value of the correlation test, and determine whether $H_0$ can be rejected (i.e., filled square: $p < 0.01$; empty square: $0.01 \leq p < 0.05$; empty circle: $p \geq 0.05$). The ordinate of a point represents the value of Kendall’s $\tau$ coefficient.

For $PTW(p,t)$ we reject $H_0$ at 0.01 confidence level for all activity types except lib. Due to insufficient projects that contain lib activities (only 15), $H_0$ cannot be rejected for this activity type even at 0.05 confidence level. We observe the strongest correlation for the four main activity types, 110n ($\tau = 0.77$), devdoc ($\tau = 0.64$), build ($\tau = 0.60$), and code ($\tau = 0.50$). This suggests that as more authors are involved in the projects, the project workload is higher in these activity types.

Projects with more authors correspond to more absolute workload in 110n, devdoc, build, and code than in other activity types.

For $RPTW(p,t)$ we again reject $H_0$ at 0.01 confidence level for all activity types except lib, for which $H_0$ cannot be rejected even at 0.05 confidence level. As opposed to $PTW(p,t)$, correlation is now negative for all activity types except devdoc and 110n, and is low for all activity types. 110n shows the strongest positive correlation (0.35), suggesting that as more authors are involved in a project, it is the share of the workload in 110n that is the highest most relative to the workload in other activity types. The positive correlation for devdoc is also due to the authors contributing to 110n, since as they perform 110n activities, they often also update the ChangeLog, which is part of devdoc. This observation generalises that of German [57], who reports similar co-updates of the ChangeLog for Evolution, one of the GNOME projects.

Projects with more authors correspond to higher fractions of workload in 110n rather than other activity types.
7.3. GOAL 1: HOW DOES WORKLOAD VARY ACROSS PROJECTS?

(a) As more authors become involved in the projects, the workload increases the most in l10n, devdoc, build, and code (black). In terms of shares, it is the workload in localisation that increases the most relative to those in other activity types (gray).

(b) As more authors become involved in the projects, they mostly contribute to l10n, devdoc, build, and code (black). The percentage of developers involved in l10n does not decrease as more developers become involved in the projects (gray).

Figure 7.11: Correlations between NAP, PTW, and PTI.
To what extent does project community size relate to the involvement (share) of authors in different activity types?

We wish to understand whether the number of authors $NAP(p)$ involved in project $p$ influences how the authors become involved in a particular activity type, in terms of their absolute involvement $PTI(p,t)$ and their involvement share $RPTI(p,t)$ for this project.

To answer the question we compute Kendall correlation between $NAP(p)$ and each of $PTI(p,t)$ and $RPTI(p,t)$, for all activity types $t \in T$. Figure 7.11b (page 131) visually summarises the results of the correlation tests for $PTI$ (drawn in black) and $RPTI$ (drawn in gray), with the same conventions as in the previous question.

For $PTI(p,t)$ we reject $H_0$ at 0.05 confidence level for lib, and at 0.01 confidence level for all other activity types. Similarly to the previous question, we observe the strongest correlation (now higher) for the four main activity types, l10n ($\tau = 0.88$), devdoc, and build ($\tau = 0.81$), and code ($\tau = 0.70$). This suggests that as more authors are involved in the projects, they are involved mostly in these activity types.

As more authors are involved in a project, they tend to be more involved in translation rather than coding.

For $RPTI(p,t)$ we again reject $H_0$ at the same confidence levels, and observe negative correlation for all activity types. lib shows now the strongest correlation ($\tau = -0.88$), suggesting that as more authors are involved in a project, the share of authors involved in this activity type is lower. This confirms that lib is the smallest activity type, and that it is performed by a limited number of developers. In addition, l10n shows the lowest correlation ($\tau = -0.09$), leading us to the following conclusion:

The number of authors involved in a project is not related to the share of authors involved in localisation activities.

Summarising the preceding discussions of Section 7.3.4, we observed the following relations between project characteristics and project specialisation. The more specialised a project, the less activity types are present in it. As more authors are involved in a project, they tend to be mostly translators and they generate a higher workload for the activity type l10n. However, we have found no evidence that higher project workload or larger project community are correlated to the overall specialisation values.
7.4. GOAL 2: HOW DOES WORKLOAD VARY ACROSS AUTHORS?

7.3.5 Summary for Goal 1

Our first research goal consisted in understanding how workload varies across projects belonging to the same ecosystem, taking into account the different types of activities performed within these projects.

First, we observed that project activity across the ecosystem follows a log-normal distribution. Next, we investigated what project properties are correlated to the project activity, and we found such a correlation for the number of activity types in which the developer community participates, and for the size of the community. Specifically, a project having a high number of activity types or having a large developer community is more active than a project having a small number of activity types or a small developer community.

By focusing on different activity types we observed that coding, development documentation, localisation, and build are the four most important ones, at the ecosystem level as well as at the level of individual projects. It is also these four activity types that attract most of the authors involved in projects. However, while it is coding that concentrates most of a project’s workload, localisation and development documentation attract most of the contributors.

Finally, we observed that most projects concentrate their workload in few activity types. We investigated the factors associated to this specialisation and found evidence that highly specialised projects are also projects including few activity types. Moreover, as projects contain more contributors, they are more commonly translators rather than coders.

7.4 Goal 2: How does workload vary across authors?

Our second research goal consists in understanding how workload varies across authors belonging to the same community. In order to address this goal we study cross-author variation of measurable author-level properties (e.g., author workload $AW$, number of projects $NPA$ in which an author is involved, number of activity types per author $NTA$) by answering the following research questions in each of the next subsections:

1. How does workload vary across authors?
2. Which kind of authors are more active?
3. How specialised are authors towards different activity types?
4. What are the characteristics of specialised authors?
7.4.1 How does workload vary across authors?

We start by studying the variation of the author workload $AW(a)$ across all projects and activity types (Figure 7.12a). As in the case of the project workload, distribution of the author workload is heavy-tailed and does not follow a power law: the $p$-value equals 0.0499 and does not exceed the recommended threshold of 0.1 [35]. As opposed to the project workload, hypothesis of log-normal distribution of the author workload can be rejected since the $p$-value corresponding to the bootstrapped Kolmogorov-Smirnov test [147] is lower than $2.2 \times 10^{-16}$.

![Figure 7.12: Distribution of author workload.](image)

(a) Distribution of author workload $AW$ (b) Approximately half of the authors is heavy-tailed but does not follow a performed less than 14 touches to power law or log-normal distribution. GNOME files.

Most authors have low workload. Few authors have high workload.

The heavy tail of the distribution of $AW(a)$ suggests a more refined analysis. To mitigate the potentially confounding effect of size, we distinguish between authors with low activity (occasional contributors) and authors with high activity (frequent contributors). Specifically, based on their $AW(a)$ values we apply equal-frequency binning and split the authors into two groups: $AW < 14$ and $AW \geq 14$. Figure 7.12b displays the breakdown of authors after binning.

Approximately half of the authors performed less than 14 file changes ($\log 14 \simeq 2.64$). In contrast, the most active author performed 185,874 file changes ($\log 185874 \simeq 12.13$).
This conclusion is concurrent with the observation by Neary and David [123] that the 40 most active developers have made 31% of all changes, while the most prolific 5% of developers have made 65% of all changes.

### 7.4.2 Which kind of authors are more active?

Are authors that participate in more activity types more active?

We first investigate whether the number of activity types $NTA(a)$ an author $a$ contributes to across the ecosystem is related to her total workload $AW(a)$. As a result of Kendall’s correlation test we confidently reject $H_0 (p < 2.2 \times 10^{-16})$ and observe a strong correlation between $NTA(a)$ and $AW(a)$ ($\tau = 0.737$).

(a) Observed linear relation between $NTA(a)$ and $\log AW(a)$ (regression line drawn in red).

(b) Residual plot.

Figure 7.13: Correlation between $NTA$ and $AW$.

The scatter plot of Figure 7.13 suggests a linear relation between $NTA(a)$ and $\log AW(a)$. We obtain the following linear regression model: $\log AW(a) = 0.69326 \cdot NTA(a) + 0.47786$, with $R^2 = 0.7971$. The fitted linear model is adequate: $F$-statistic equals 20030 on 1 and 5146 degrees of freedom with the corresponding $p$-value not exceeding $2.2 \times 10^{-16}$, $p$-values for the coefficient and the intercept do not exceed $2.2 \times 10^{-16}$. The points in the residual plot appear randomly dispersed around the horizontal axis. We conclude that the author activity increases exponentially (due to the use of $\log AW$ in the formula) as authors contribute to more activity types. Increasing the number of activity types by one doubles the effort ($e^{0.69326} \approx 2$). Figure 7.13 also reveals that 1452 (28%) authors are involved in a single activity type. In Section 7.4.3 we investigate in which activity types these authors specialise themselves.
The more activity types an author participates in, the more active she is: increasing the number of activity types by one doubles the workload.

Are authors that contribute to more projects more active?

How does the number of projects $NPA(a)$ an author $a$ is involved in correlate to the total workload $AW(a)$ for that author? As a result of the Kendall correlation test we confidently reject $H_0$ ($p < 2.2 \times 10^{-16}$), and observe above average correlation ($\tau = 0.573$). This suggests that the author workload increases as authors become involved in more projects. We do not describe the relation between $NPA(a)$ and $AW(a)$ further since we could not obtain linear regression models for which the points in the residual plot would appear randomly dispersed around the horizontal axis, hence the linear models were not appropriate for the data.

The more projects an author contributes to, the more active she is.

7.4.3 How specialised are authors towards different activity types?

To what extent are authors specialised in few activity types?

We intuitively expect that authors are mostly specialised in few activity types, similar to what we observed for the specialisation of projects in Section 7.3.3. The specialisation $RAWS(a)$ of an author $a$ can be interpreted as how this author specialises her relative workload towards few activity types. It is computed by applying the Gini index to aggregate the $RATW(a,t)$ values over all types $t \in T$. Figure 7.14 displays the variation across authors of $RAWS(a)$, for the entire ecosystem community as well as for each of the two groups obtained after binning.

Note that different authors contribute to different activity types (e.g., not all authors contribute to test activities). Since we are interested in comparing specialisation for different authors, we consider for each author the set of all possible activity types (i.e., we do not ignore the activity types $t$ for which $RATW(a,t) = 0$ when computing $RAWS(a)$).

Using the same equal-frequency binning for $AW$ as in Section 7.4.1, we observe a clear distinction between the specialisation of occasional ($AW < 14$) and frequent ($AW \geq$
14) contributors. Since they contribute very few changes in total to the ecosystem, the occasional contributors are very specialised, more than the frequent contributors: the median for the occasional contributors group equals 0.9285, which is also the maximal value of the Gini index for populations of size 14, i.e., $1 - 1/14$. By definition of the Gini index it follows that most of the occasional contributors participate in a single activity type. Note that the double usage of the value 14 is coincidental: in “$\text{AW} < 14$”, 14 was the threshold found for $\text{AW}$ as a result binning, while in “$1 - 1/14$”, 14 refers to the number of activity types.

![Figure 7.14: Relative author-workload specialisation $\text{RAWS}(a)$.](image)

Our observation that the occasional contributors are specialised more than the frequent contributors is supported by statistical tests. The relative effect for (frequent, occasional) is 0.815 and the corresponding $p$-value is too small to be computed exactly. Since the relative effect exceeds 0.5, the specialisation values for the occasional contributors tend to be larger than those for the frequent contributors. The Wilcoxon-Mann-Whitney test allows us to derive the same conclusion, $p < 2.2 \times 10^{-16}$.

Even though less specialised, the frequent contributors also display a very high median of $\text{RAWS}(a)$ (0.82), even higher than the corresponding median of $\text{RPWS}(p)$ from Goal 1 (0.77, Figure 7.8). Therefore, there is high inequality in the distribution of workload across the different activity types for most of the frequent authors. Overall, we conclude that most of the authors’ workload is concentrated in few activity types, while the remaining activity types only account for a small fraction of the workload.

While contributing to different projects within the ecosystem, most authors concentrate their workload in few activity types. Moreover, occasional contributors typically participate in a single activity type.
To what extent are authors specialised towards different activity types?

**Workload** The specialisation of an author \( a \) towards a certain activity type \( t \) can be expressed as the specialisation of her relative workload \( RATW(a, t) \), i.e., the total number of file touches that \( a \) performed for activity type \( t \) across the ecosystem relative to the total number of file touches that \( a \) performed for all activity types across the ecosystem. High \( RATW(a, t) \) values reflect that most of the workload of \( a \) across the ecosystem is directed towards activities of type \( t \), i.e., \( t \) is a predominant activity type for \( a \). Low \( RATW(a, t) \) values reflect that activities of type \( t \) are but auxiliary for \( a \).

Figure 7.15 (top left) depicts the overall variation across authors of \( RATW(a, t) \) for each activity type \( t \), for all authors. As before, for each boxplot we only consider the authors that contribute to \( t \) (i.e., the workload \( ATW(a, t) > 0 \)), and display their number below each activity type. We observe the same outstanding activity types as in Figures 7.7 and 7.9, namely code, devdoc, and l10n. Specifically, we observe that the third quartile for l10n coincides with 1, i.e., approximately 25% of the translators (526 out of 2008) focus exclusively on l10n, corresponding to slightly more than 10% of the entire GNOME community. A similar finding has been reported for KDE by Robles et al.[139].

The \( \tilde{T} \)-graph on the right confirms that code is the dominant activity type, while db and lib have the smallest values. Therefore specialisation of authors towards certain activity types follows specialisation of projects, since most authors specialise in the four previously observed main activity types. The predominance of code and l10n is also recognised by members of the ecosystem community in mailing list discussions:

“[...] gnome *is* a code-centric organisation. The coders are our sine qua non—without them, we have nothing. Translators are probably a close second to that—without them, we have no international coders. Past that, no group of people in the project are indispensable to the current state of the project, or even close to it.” Villa [175]

Most authors specialise in code, l10n, and devdoc activities. Among those activity types, code is predominant.

Using equal-frequency binning we can obtain more fine-grained information for the occasional contributors (\( AW < 14 \)) and the frequent contributors (\( AW \geq 14 \)). We displayed both groups in the middle and bottom boxplots of Figure 7.15. Visual comparison of these sets of boxplots reveals a clear distinction in behaviour for the l10n activity:
Figure 7.15: Boxplots for $\text{RATW}(a,t)$ per activity type $t$. Zero values are excluded. By definition of $\text{RATW}(a,t)$ ($\text{AW}(a)$ appears in the denominator), the lower whiskers in the $\text{AW} < 14$ boxplots cannot be lower than $1/13$. For $\text{AW} < 14$ the $\mathbf{T}$-graph does not include $\text{db}$ and $\text{lib}$ since the $\mathbf{T}$ procedure is not applicable for groups of size one.
in the $AW < 14$ case, the median for $RATW(a,110n)$ is 1, while in the $AW >= 14$ case it is around 0.1. Indeed, statistical tests show that occasional contributors are specialised in localisation more than the frequent contributors. The relative effect for (frequent,occasional) is 0.907 and the corresponding $p$-value is too small to be computed exactly. Since the relative effect exceeds 0.5, the $RATW(a,110n)$ for the occasional contributors tend to be larger than those for the frequent contributors. The Wilcoxon-Mann-Whitney test allows us to derive the same conclusion, $p < 2.2 \times 10^{-16}$.

Since the overall median for $RATW(a,110n)$ is also relatively small ($\approx 0.3$), the preceding discussion suggests that occasional contributors prefer to specialise in localisation rather than other activity types. This observation is supported by the T-graphs, in which we observe an inversion of the relation between $110n$ and code from $AW < 14$ to $AW \geq 14$: while code is the dominant activity type for frequent contributors, $110n$ is the dominant one for occasional contributors. On the other hand, the relations between code and devdoc, and between code and build are consistent.

For frequent contributors, devdoc and build remain the other two predominant activity types. For occasional contributors, although img or config might appear visually to have higher values, the data does not provide enough evidence to support this observation using the T-procedure (see T-graph).

Frequent contributors tend to specialise in the code activity, while occasional contributors tend to specialise in the $110n$ activity. For both types of contributors devdoc and build remain important activities.

A similar difference in behaviour between occasional and frequent GNOME contributors with respect to code and $110n$ has also been observed by Neu et al. [124].

The authors assume persons “who contributed a lot but only to a relatively small number of projects” to be coders (i.e., the people located under a logarithmic-like curve in Figure 7.16a), while those “who committed less often but to more projects” to be translators (i.e., the people placed above an exponential-like curve in Figure 7.16a). While this classification is only qualitative, it is confirmed by our quantitative analysis (Figure 7.16b). In our case the y-axis also corresponds to the number of projects per author $NPA(a)$, while the x-axis corresponds to the author workload $AW(a)$ – expressed as number of file touches per author, so more fine-grained than the number of commits per committer used by Neu et al. [124]. In the plot we overlay per author $a$ the $RATW(a,\text{code})$ values (blue crosses) and $RATW(a,110n)$ values (red squares), where the size of each symbol encodes the $RATW$ value. Our results are consistent with those of Neu et al.[124]: coders are typically very active contributors involved in a relatively small number of projects,
7.4. GOAL 2: HOW DOES WORKLOAD VARY ACROSS AUTHORS?

(a) Visualisation by Neu et al. [124]: each square depicts a committer; the number of projects is encoded both on the y-axis and in the colour of each square; the size of a square corresponds to the lifetime in days of a committer within GNOME; ⓣ: translators, ⓦ: developers, ⓧ: outlier, ⓩ: no man’s land. Persons under the logarithmic-like curve are assumed to be coders, persons above the exponential-like curve are assumed to be translators.

(b) Our own visualisation.

Figure 7.16: Visualisations of occasional and frequent contributors.
while translators are less active but are involved in more projects. Examples of potential misclassifications following the qualitative approach include developer C, a coder with low activity but involved in many projects, or developer B, a contributor active in both code as well as l10n, having high activity but involved in relatively few projects. Mixed patterns of involvement in code and l10n (for which developer A is an example) are in-line with the following excerpts from the mailing list discussions: while translators do not typically code, some start out with just translating, but continue with fixing bugs and then coding.

"Furthermore, in GNOME, we have many translators that started out with just translating, but continued with fixing bugs, and some are full time coders now. We should be proud of this integration." [145]

"As you have pointed out yourself, translators are usually not hackers/ - coders." [144]

**Involvement** The relative author involvement $RATI(a,t)$ is defined as the number of projects in which author $a$ performs activities of type $t$ relative to the total number of projects in which she is involved. High $RATI(a,t)$ values reflect that in most of the projects author $a$ is involved in, she contributes to activities of type $t$. Similarly, low $RATI(a,t)$ values reflect that $a$ performs activities of type $t$ only sporadically, i.e., she only performs activities of type $t$ in a small fraction of the projects she is involved in.

For all authors, the variation of $RATI(a,t)$ per activity type $t$ is illustrated in Figure 7.17 (top). Zeros are again ignored. The median value of 1 for the code, devdoc, and l10n activity types signifies that, once authors are involved in these activity types, they perform the same activities in all projects they contribute to. Less pronounced is the recurrence of authors in build, config, and doc activity types, for which there is more spread. Even though these activity types are common in software projects, they are less specialised and thus can be performed by different authors in different projects. As expected, the lowest median values correspond to the activity types least common to the software projects considered, i.e., lib and db (both leaves in the $T$-graph). Since the boxplot for $RATW(a,db)$ from Figure 7.15 is very low, we conclude that authors who prefer to specialise in db activities contribute to other activity types in the projects in which db activities are absent.

Most authors contributing to code, devdoc, and l10n activity types in one project, do so in all other projects they contribute to. In contrast, database developers “wear many hats”, i.e., they contribute to other activity types in projects where db is not available.
Figure 7.17: Boxplots for $RATI(a, t)$ per activity type $t$. Zero values are excluded. For $AW < 14$ the $T$-graph does not include db and lib since the $T$ procedure is not applicable for groups of size one.
To illustrate the point of the versatility of database developers we mention that one of the most active database contributors in *anjuta* has been involved in *gdl* as coder, translator, builder, and even UI designer.

To obtain additional insights we study the difference between occasional \((AW < 14)\) and frequent \((AW \geq 14)\) contributors in the middle and bottom boxplots of Figure 7.17. Visual comparison of both sets of boxplots reveals striking differences: in the \(AW < 14\) case, all activity types except \(db\) have a median of 1, suggesting that once occasional authors are involved in these activity types, they perform the same activities in all projects they contribute to. This should not come as a surprise: further inspection of the data revealed that 77.3% (1991 out of 2576) of the occasional contributors only participate in a single project.

On the other hand, the results (and \(\tilde{T}\)-graphs) for \(AW \geq 14\) are similar to those for the entire ecosystem community: code, devdoc, build, and l10n are again the predominant activity types, suggesting that authors involved in these activity types choose to specialise in them in all projects they contribute to. For example, we have identified a frequent contributor \((AW = 1459)\) involved in 28 projects, who dedicates more than 94% of his effort to code.

Most occasional contributors participate in a single project. Frequent contributors specialise in code, devdoc, and to a lesser extent build and l10n, i.e., they perform these activities in most projects they participate in.

### 7.4.4 What are the characteristics of specialised authors?

We wish to understand which of the author characteristics studied previously (i.e., author workload \(AW(a)\), number of activity types per author \(NTA(a)\), and number of projects per author \(NPA(a)\)) are related to her degree of specialisation.

**Which characteristics of an author are observed when she is specialised towards few activity types?**

In Figure 7.14 we observed that most authors specialise their workload in few activity types. We expect that authors contributing to many activity types prefer to spread their work across these types rather than concentrate it in few of them. Thus, we expect that
7.4. GOAL 2: HOW DOES WORKLOAD VARY ACROSS AUTHORS?

authors involved in many activity types, as well as authors involved in many projects tend to be less specialised.

To answer the question we study Kendall correlation between $RAWS(a)$ and each of the author-specific $AW(a)$, $NTA(a)$, and $NPA(a)$ metrics. For all three metrics, we confidently reject $H_0$ at 0.01 significance level ($p$-value $< 2.2 \times 10^{-16}$). Similarly to the complementary question in Section 7.3.4, we observe negative correlation for all three metrics. However, now the correlation is much stronger: $\tau = -0.342$ for $NPA(a)$, $\tau = -0.497$ for $AW(a)$, and $\tau = -0.751$ for $NTA(a)$.

Three factors (i.e., number of activity types, number of projects, and number of file touches) are negatively correlated to specialisation of authors: the higher a factor, the less specialised an author is towards few activity types.

To what extent does the number of projects an author is involved in relate to her workload (share) for a particular activity type?

We wish to understand whether the number of projects $NPA(a)$ author $a$ is involved in correlates to her workload $ATW(a,t)$ and her relative workload $RATW(a,t)$ for a particular activity type $t$. We expect that authors that participate in many projects contribute to l10n rather than code activities, hence the more projects an author contributes to, the higher the workload in l10n should be.

To answer the question we compute Kendall correlation between $NPA(a)$ and $ATW(a,t)$ on the one hand, and between $NPA(a)$ and $RATW(a,t)$ on the other hand, for all activity types $t \in T$. In concordance to Section 7.4.3, we discard the authors for which $ATW(a,t) = 0$. Figure 7.18a visually summarises the results of the correlation tests for $ATW$ (black) and $RATW$ (gray), using the same visual conventions as in Figures 7.11a and 7.11b.

For both $ATW(a,t)$ and $RATW(a,t)$ we confidently reject $H_0$ at 0.01 confidence level for all activity types except lib. In case of $ATW(a,t)$ correlation is positive for all activity types (except lib which is statistically insignificant), suggesting that the workload of authors increases in all activity types as they contribute to more projects. The four main activity types we previously observed at project level are therefore also confirmed at author level, with the highest correlation being observed for l10n ($\tau = 0.58$) and devdoc ($\tau = 0.57$) activity types. In contrast, correlation is negative for all activity types in case of $RATW(a,t)$ (e.g., $\tau = -0.33$ for code and $\tau = -0.17$ for l10n) The statistical
analysis therefore supports the observation one can make by inspecting Figure 7.16: the upper two-thirds of the picture are dominated by large red symbols (= l10n), while blue symbols (= code) in this region remain small and barely visible.

As authors are involved in more projects, they contribute to l10n rather than code activities.

To what extent does the number of projects an author is involved in relates to the share of projects in which she performs a particular activity type?

We wish to understand whether the number of projects $NPA(a)$ an author is involved in relates to the absolute author involvement $ATI(a,t)$ or the involvement share $RATI(a,t)$ of these projects in which she performs activities of a particular type. We expect that not all activity types can support the same growth in terms of the number of projects in which they are performed. For example, we expect that the (relative) number of projects in which an author contributes to db or lib activities does not increase significantly as the author is involved in more projects since these activity types are performed in few projects in total. Moreover, even for main activity types such as l10n and code, we expect that it is more common for authors to perform l10n rather than code activities in most of the projects they are involved in, as this number grows.

To answer the question we compute Kendall correlation between $NPA(a)$ and $ATI(a,t)$ on the one hand, and between $NPA(a)$ and $RATI(a,t)$ on the other hand, for all activity types $t \in T$. Figure 7.18b visually summarises the results of the correlation tests for $ATI$ (drawn in black) and $RATI$ (drawn in gray), for all activity types, under the usual conventions. For both $ATI(a,t)$ and $RATI(a,t)$ we confidently reject $H_0$ at 0.01 confidence level for all activity types except lib. For $ATI(a,t)$ we observe the strongest correlation for code and devdoc ($\tau = 0.85$), build ($\tau = 0.76$), and l10n ($\tau = 0.72$), while db exhibits the lowest correlation among the statistically significant activity types ($\tau = 0.33$). For $RATI(a,t)$ we observe negative correlation for all activity types (e.g., $\tau = -0.63$ for code, and $\tau = -0.35$ for l10n). This indeed confirms our expectation. Recall that $RATI(a,t)$ is defined as the ratio between $ATI(a,t)$ and $NPA(a)$. Therefore, although increases in $ATI(a, code)$ and $ATI(a, l10n)$ both match increases in $NPA(a)$ (high positive correlation), it is only $RATI(a, code)$ that decreases as $NPA(a)$ increases (high negative correlation). Therefore, we can say that the percentage of projects in which an author participates for the l10n activity does not decrease as she is involved in more projects.
7.4. GOAL 2: HOW DOES WORKLOAD VARY ACROSS AUTHORS?

Authors that are involved in more projects tend to participate more in l10n for these projects than in code, i.e., the fraction of projects in which an author participates in l10n does not decrease as she is involved in more projects. (Recall the complement: as projects attract more developers, these are more commonly translators rather than coders).

Our finding concurs with the observation of Jergensen et al. [82] made for a subset of six GNOME projects. The overlap between the committer communities of these projects increases when documentation and translation committers are included as opposed to source code committers only. We conjecture that the ease of participation in cross-project translation activities is fostered by Damned Lies, the Web application used to manage the localisation of GNOME\textsuperscript{4} (cf. description of intltool, one of the predecessors of Damned Lies in [155, p. 44]).

\textsuperscript{4}l10n.gnome.org
(a) As they are involved in more projects, more authors concentrate their workload in \textit{l10n} and \textit{devdoc} than in \textit{code} and \textit{devdoc} in their new projects other activity types (black). In addition, the share of their workload in \textit{code} decreases in comparison to \textit{l10n} (gray).

(b) As they are involved in more projects, more authors contribute to \textit{code} and \textit{devdoc} in their new projects than in \textit{l10n} (black). In addition, the share of their projects in which they \textit{code} decreases in comparison to \textit{l10n} (gray).

Figure 7.18: Correlation between \textit{NPA}, \textit{ATW}, and \textit{ATI}. 
7.5. THREATS TO VALIDITY

7.4.5 Summary for Goal 2

Our second research goal consisted in understanding how workload varies across authors belonging to the same community, taking into account the different types of activities they perform.

First, we observed that author activity across the community follows a heavy-tailed distribution: most authors are occasional contributors with little activity, while relatively few authors are frequent contributors that are very active. Specifically, approximately half of the authors performed less than 14 file touches, while the most active author performed 185,874 file touches.

Next, we investigated the factors that influence how active authors are. We found that the more activity types an author participates in, or the more projects she contributes to, the more active she is. Moreover, we observed that when contributing to different projects within the ecosystem, authors prefer to specialise in a small number of activity types. In particular, occasional contributors typically participate in a single project, and a single activity type.

Focusing on different activity types, we observed that coding, development documentation, localisation, and build are also the 4 activities in which members of the ecosystem community prefer to specialise. While frequent contributors prefer coding, occasional ones specialise in localisation. Both frequent as well as occasional contributors are attracted to development documentation and build.

In terms of their versatility across projects, we observed that most authors contributing to coding, development documentation, and localisation in one project, do so as well in other projects they contribute to within the ecosystem. However, as authors become involved in more projects, it is more common for them to participate in localisation rather than coding across the different projects. On the other hand, authors contributing to scarce activities (such as database) “wear many hats”, i.e., they contribute to other activity types when their preferred ones are not available.

7.5 Threats to validity

As in any empirical study, there are many potential threats to validity in our research.
CHAPTER 7. ON THE SPECIALISATION OF GNOME WORKLOAD

Construct validity seeks agreement between a theoretical concept and a specific measuring device or procedure. In this respect, we equated the notion of contributor to the notion of Git author in our study. By taking into account other data sources (e.g., mailing lists and bug trackers), as has been done in Chapter 6, we could consider a larger set of GNOME contributors and activity types, which could affect our results. Another potential threat is the lack of agreement between the theoretical concept of a “author” and a specific author identification technique described in Section 7.2.2. As discussed in Chapter 4, identity matching can never be perfect due to the presence of false positives and false negatives. Using a wide portfolio of complementary algorithms we have reduced these to a minimum. In addition, we manually checked and corrected the remaining problems. Even if some incorrect identity matches may remain, their number will be limited and will not significantly influence the results presented in this chapter. Note that we used a threshold of 0.8 for the similarity measures used during identity matching. This threshold was chosen based on a limited number of tests but is compatible with the obtained results of the study we achieved in Chapter 4.

Construct validity might also have been affected by our operationalisation of an author’s activity. Our activity identification builds on and extends the work of [139]. We stress, however, that changing the rules (regular expressions) and the order in which they are evaluated may lead to different results. Looking at the exact changes made to each file may lead to a more precise identification of the activity type. In addition, as discussed in Section 8.1, our approach does not allow to associate more than one activity type to the same file.

Construct validity is also related to our definition of workload and involvement as proxies for the actual development effort. To determine the workload we counted the number of file touches by an author for a project, without taking into account the size of the file change or the effort that was needed for making such a change. In considering the number of file touches rather than the modification size we follow a popular approach in software evolution research [39, 167]. A similar proxy of the development effort has been used by German [55].

Internal validity is related to validity of the conclusion within the experimental context of GNOME. A first threat is that we were not able to extract and analyse data from all 1358 GNOME projects (i.e. 97%). We have left out 42 projects (3%) due to data extraction errors, but given the low percentage this will have little influence on the validity of the results. Since our study did not involve repeated application of a treatment, typical threats to internal validity such as history, maturation, or mortality [181] could not have affected the results of our study. Furthermore, we have paid special attention to the appropriate use of statistical machinery [150] (cf. Sections 7.2.5 and 7.2.6).
**7.6. RELATED WORK**

*External validity* is the validity of generalisations based on this study beyond GNOME. External validity is of no importance for this study as no claims are made about the generalisability of our results to other ecosystems. Although the studies presented in this chapter can be replicated on other open source ecosystems\(^5\), the obtained results may vary, as each ecosystem has its own specific community and process. For instance, the significant share of translation activities in GNOME might be related to the special GNOME Live! translation project or Damned Lies, the Web application used to manage the localisation of GNOME.

### 7.6 Related work

In addition to the state of the art detailed in Chapter 2, this section presents research studies related to the GNOME activities. Identity matching approaches are discussed in Chapter 4. In this chapter we have studied the relationship between projects, authors and activity types in the open source software ecosystem GNOME. Throughout the chapter we have added references to related work pertaining to individual steps in our analysis process. For example, existing work related to data analysis is discussed in Sections 7.2.5 and 7.2.6. In Sections 2.2 and 7.6.1 we discuss existing results related to studies of developers and their activities.

#### 7.6.1 Studies of open source software contributors

Many researchers have investigated the roles developers play in open source software projects. Capiluppi et al. [23] attempted to characterise open source projects, their evolution, and the developer communities responsible for their maintenance, by studying 400 projects hosted by the FreshMeat\(^6\) portal. While they distinguish between *stable* and *transient* developers based on the amount of changes they perform, we classify developers based on the types of files they touch. In addition, the projects analysed by them are not necessarily related, hence could be maintained by independent authors. In contrast, GNOME community members participate in multiple projects across the GNOME ecosystem. Shibuya and Tamai [151] also distinguish between frequent and occasional contributions, based on the number of commits developers contribute each month. However, even though they distinguish between different *activities* related to involvement in

---

\(^5\)We have provided a replication package here: [www.win.tue.nl/mdse/gnome](http://www.win.tue.nl/mdse/gnome). However, it will first need to be adapted in order to be applicable to other software ecosystems.

\(^6\)http://freecode.com
a project (e.g., participating in mailing lists, reporting bugs, developing), they do not
distinguish between different development activities (e.g., coding, testing, writing docu-
mentation).

Mockus et al. [118] performed two case studies on the Apache and Mozilla projects
where they investigated the roles and responsibilities of developers. Their approach dis-
tinguishes between developers who contributed code submissions, performed bug fixes, or
reported problems. Although they do not study differences between development activity
types, they observe specialisation of contributors towards a single role. Our data shows
similar features: approximately 20% of the developers involved in code activities, and
25% of the developers involved in localisation activities do not contribute to other activ-
ity types. Similarly to Mockus et al., Nakakoji et al.[121] distinguish between developers,
bug fixers, and bug reporters. Furthermore, they have proposed a more refined classifi-
cation of developer kinds: peripheral developers, active developers, core members, and
project leaders. The “onion model” proposed suggests that there are more core members
than project leaders, more active developers than core members, more peripheral devel-
opers than active ones, etc. Similar hypotheses have been studied by Dinh-Trong and
Bieman [46]. The Open Source community itself recognises that developers play different
roles as witnessed, e.g., by recording “credited developers” and “maintainers” as opposed
to uncredited developers or maintainers in Linux [119].

Benefits of incorporating the more refined classification in our work include discovering
whether persons classified as core members in a number of projects tend to limit their
involvement in other projects to bug reporting. However, as shown by Poncin et al. [133],
integration of the refined classification proposed by Nakakoji et al. necessitates additional
analysis of bug tracker information.

7.6.2 Studies of the GNOME ecosystem

GNOME was part of the MSR 2009 and 2010 Mining Challenges. In 2009, there was
a general challenge to demonstrate the usefulness of mining tools on the GNOME case
study, and a prediction challenge to predict the code growth at project level.

Casebolt et al. [30] found an inverse relation between file size and the notion of author
entropy, suggesting that large files are more likely to have a dominant author than small
files. The notion of author entropy characterises the distribution of author contributions
to a file, and is therefore related (at least in spirit) to our use of inequality indices such as
Gini or Theil. The main difference is that we did not focus on the author collaboration
for individual files.
Similarly to Neu et al. [124] we recognise the importance of combining the analyses of the ecosystem and an individual project, as well as the community and an individual contributor. However, while Neu et al. focused on visualisation of GNOME data based on a number of assumptions, we conducted statistical analyses. Our findings support their assumptions, since we also observed that persons “who contributed a lot but only to a relatively small number of projects” are typically coders, while those “who committed less often but to more projects” are typically translators.

In their study of effort, co-operation and co-ordination in GNOME, Koch and Schneider [88] have observed significant differences between contributions of different developers in terms of lines of code. Both their data and our data on the workload in terms of file touches (Section 7.3.1) show similar features: the distributions are left-skewed and the maximal value is more than an order of magnitude larger than the mean, i.e. both distributions are heavy-tailed [162]. A similar distribution seems to be suggested by partial data on percentages of modification requests per developer, as reported by German [55]. As opposed to our work, Koch and Schneider consider a more advanced approach to effort estimation that takes into account lines of code added or deleted as well as the communication between the developers via mailing lists. Furthermore, while Koch and Schneider follow a holistic approach, we augment such results with more fine-grained analyses of individual GNOME projects. Similarly to Koch and Schneider, Gousios et al. [67] developed an advanced measure of individual developer contribution based on information from the source code repository, the mailing lists and the bug tracking systems, and applied the measure to a number of GNOME projects. Jergensen et al. [82] have studied six GNOME projects in order to understand how developers join, socialise and develop within GNOME. They observed, among others, that very experienced developers are less involved in the actual coding. Similarly to our work, Jergensen et al. have observed that translation and documentation are more cross-project activities than coding.

Summarising this discussion we observe that most of the studies so far either followed a holistic approach and considered GNOME as one system, or focused on a number of example GNOME projects such as Evince or Nautilus.

Finally, Lopez Fernandez et al. [102] studied relations between the GNOME developers by means of social network analysis, while Ernst and Mylopoulos [51] studied perception of software quality requirements in some of the GNOME projects.

7.7 Conclusion

In this chapter we studied the workload variation of the projects belonging to an open source software ecosystem, and the workload variation of the contributors belonging to
the ecosystem community. To achieve this, we used a portfolio of statistical techniques to carry out a case study on GNOME, a large and well-known open source ecosystem and associated community (with over 1,300 different projects and over 5,000 active authors).

By analysing the GNOME mailing list archives, we observed that GNOME contains both paid contributors and volunteers, and it can be expected that their workload is different. The GNOME community also acknowledges that, while coding is the most important activity, other activities such as translation/localisation are indispensable. In addition, translators are usually not coders, and most of the other GNOME activity types (such as documentation, user interface design, etc.) are considered to be less important.

To confirm these informal observations, we studied the GNOME ecosystem with two research goals in mind: to understand how workload varies across projects and to understand how workload varies across contributors. To achieve this, we defined a novel set of metrics, parameterised by project, author and activity type, with coding, localisation and development documentation being the most important activity types. This set of metrics can be considered as a contribution in its own, as they can be reused easily for studying other software ecosystems. Of particular importance are the specialisation metrics that are defined based on the Gini inequality index. An additional contribution consists in introducing $\tilde{T}$-graphs, a novel approach to reporting the results of the $\tilde{T}$ procedure.

Concerning the first research goal, we observed that project workload across the ecosystem follows a log-normal distribution. Looking into more detail, we found two characteristics that positively correlated to the project workload: the size of the project community and the number of activity types contained in the project. The workload was mainly concentrated in four activity types, that also attracted most of the project’s contributors: coding, development documentation, localisation, and build. Of these, coding concentrates most of a project’s workload, while localisation and development documentation attract most of the contributors. We also found evidence that highly specialised projects only include few activity types. In contrast, we did not find evidence that projects with more activity types or larger communities are less specialised.

Concerning the second research goal, we observed that author workload across the GNOME ecosystem follows a heavy-tailed distribution: most contributors have little activity (approximately half of the contributors performed less than 14 file touches), while a small number of contributors have a very high workload. We found that a contributor’s workload is positively correlated to the number of projects she contributes to, as well as the number of activity types she participates in. We also observed that contributors prefer to restrict themselves to a small number of activity types. In particular, the many occasional contributors typically restrict themselves to a single project and a single activity type. While occasional contributors are mainly specialised in the localisation activity,
frequent contributors tend to prefer coding. Both kinds of contributors are also often involved in development documentation and build. Most contributors to one of these four activity types in one project, also tend to contribute to these types in the other projects they are involved in. However, the more projects a contributor is involved in, the more she tends to participate in localisation as opposed to coding.

Overall, our empirical study has allowed us to confirm that there is no such thing as a uniform ecosystem of projects and contributors: when taking into account the activity types and the workload, there is a lot of variation across projects and across contributors, but with a clear preference towards the activity types of coding, localisation, development documentation, and building. It is quite possible that other ecosystems than GNOME may reveal other activity patterns.
Part IV

Conclusion
This chapter presents the possible extension to our work. It details the improvement and the further studies that can be achieved on the basis of the tools we developed and the empirical studies we carried out. Finally, this chapter presents further works that can be undertaken on the basis of the lessons we learned.
8.1 Tooling

The identity merge algorithms we compared in Chapter 4 and we used in Part III may be improved or adapted to a particular context in several ways. Privacy issues can occur if a person involved in an open source project’s evolution does not wish to be traced back from the repositories on which he has an account. A way to avoid this would consist in replacing identity labels by numbers, but this would go at the expense of the reproducibility of the study [138].

Identity merge algorithms nearly always produce false positives, so a manual post-check is needed to refine merge results. We can try to reduce this post-treatment by taking into account project-specific or domain-specific rules and constraints. Another means to improve merge algorithms consists in taking into account timezones. Depending on the e-mail provider or e-mail application’s configuration, the mailer’s timezone may be sent simultaneously with the mail content, enabling the identification of the mailer’s timezone.

Most of the false positives are due to the fact that logins or e-mail prefixes only contain a first name. Persons with the same first name may accidentally be merged because their e-mail addresses and bug tracker accounts are considered similar. We could adapt our algorithms to avoid these unnecessary merges.

The approach described in Chapter 5 for classifying files may be considered as too restrictive for some purpose. In some cases, multiple classification would have been useful. For example, as previously explained, a file called /test/ClassTest.java could be classified as test activity type because it presumably contains unit tests as well as code because Java test files are also source code files.

Another limitation of our approach is that we classify files in a particular activity type based on the file path and file extension. This is not always sufficient. To determine if a file is really a code file or test file, for example, one would need to parse the file’s contents with analysis tools such as SLOCCount\(^1\) or CLOC\(^2\). However, a thorough analysis of files belonging to a software ecosystem is prohibitively time consuming. For more details, we refer to Zaidman et al. [184] who used open source repository mining to study the co-evolution of production code and test code. A more refined classification and treatment of files per activity type is beyond the scope of this dissertation.

\(^1\)http://www.dwheeler.com/sloc\(^\text{count}\)/
\(^2\)http://cloc.sourceforge.net/
8.2. Empirical studies

8.2.1 Pareto principle and power law

Chapter 6 presented an empirical study of the Pareto principle and the power law in OSS projects, as well as its evolution over time. A possible extension of this study could consist in using statistical tools to formally determine which projects follow the Pareto principle and/or a power law. In order to avoid a bias in our observation of the evolution of distribution inequalities, we intend to extend our study by showing a non-cumulative representation of these inequalities. However, the way old events should be ignored this value should be computed is itself subject to discussion.

8.2.2 Variation of the specialisation

The results presented in Chapter 7 can be extended in numerous ways. While in this chapter we have considered all GNOME projects as being equal, in reality they are classified under different categories (see git.gnome.org/browse/): Archived, Administration tools, Bindings, Desktop, Development tools, Infrastructure, Platform, Productivity tools, Deprecated and Other. We could also cluster GNOME projects together along
other dimensions. For example, all GNOME projects related to multimedia activity (e.g., bonobo-media would belong to this cluster). An other additional dimension could be the programming language used in source code files. We intend to look into different such classifications and clusterings to statistically investigate whether projects belonging to the same category share common properties, and to what extent differences between projects can be explained by these categories [37, 148, 149].

An important area of future work is to look at how the presented metrics and statistics evolve over time. This would allow us to detect certain trends (or trend breaks) in how ecosystems and communities evolve, predict future evolutions, and compare the evolution of projects (or authors) against one another.

In a preliminary study, we found that the specialisation of the GNOME developer community members tend increase over time. During its entire lifetime, counted in number of files touched, the majority of all GNOME effort is located in the coding activity. During its entire time life, counted in number of members involved, the coding activity is also the one involving the most of contributors, followed by the localisation activity, even though the size of the localisation community is not reflected in its effort. This was already observed in the past by German [55] who explained this by the fact that most of the localisation effort of Gnome is done by small language teams that work on a voluntary basis.

As Figure 8.1 shows, a clear decrease of the coding activity occurs in 2004. All the activity types seem to present a decreasing workload since 2009. This is particularly visible for the coding and the translation activities. Even though we don’t have formally showed a correlation between these events, 2009 is the year during which GNOME 3.0 has been announced. This is also during this year that GNOME projects massively switched to Git source code repositories. Finally, the 1.0 revision of the Damned Lies project, which the main platform for the translation of the GNOME projects, has been released in 2009. Figure 8.2 shows that there is no activity communities growing over time at the expense of the others.

We intend to take into account other data repositories in future studies. In particular, we wish integrate data coming from bug trackers and mailing lists [67, 133]. On the one hand, this gives us access to a richer source of information. On the other hand, it makes the integration of these different data sources more challenging. We also intend to apply our study to other software ecosystems than GNOME. KDE and GNU are likely (but not the only) candidates.

Distinction between different development activities, e.g., coding and translation, can be used to refine measures of recognition and experience intended for quantification and
Figure 8.1: Evolution of the yearly absolute workload per activity community, expressed as the number of file touches during the considered time period.

Figure 8.2: Evolution of the yearly relative workload per activity community, expressed as the number of file touches during the considered time period.
comparison of the contributions of open-source software developers in an objective, open, and reproducible way [28]. These measures can be used both by software developers looking for a job and by recruiters evaluating suitability of such candidates [27].

Finally, an other possible extension of the study presented in Chapter 7 is the use of more characteristics when studying the variation across projects and authors. For projects we intend to include, among others, project size, project maturity, main programming language used, and application domain. For communities, developer seniority, team size, and team structure, among others, could be taken into account.

8.2.3 Further work and studies

In this section, we present possible extensions of our work based on the lessons learned.

A first possible extension is a replication study, based on another OSS ecosystem. As explained in Section 9.2.3, we cannot generalise our observations to other OSS ecosystems. In order to assess the sensitivity of our results to the ecosystem specificities, we plan to replicate our empirical studies by considering other open source ecosystems such as KDE and OpenBSD. Such a replication is made easier by the adaptability of our application framework, but needs an important effort to be achieved. This effort is mainly due to the manual steps of the identity merging process and the interpretations of the results of statistical tests. Nevertheless, some of these interpretations may be automated.

Another further extension to our empirical studies is the analysis of developer behaviour characterised by the activity types in which the developers are involved, as well as their workload in these activity types. A preliminary study reveals that developers can be classified in four main profiles. We intend determine if developer profiles follow some evolutionary patterns.

As explained in Section 1.1, the work presented in this dissertation has as a long-term goal the building of tools that provide a relevant information and a helpful support to the communities involved in a software ecosystem. Our studies reveal that most active contributors are involved in more types of project repositories and are active in more activity types than occasional contributors. Because most active contributors are responsible of most of the changes made on the ecosystem projects, it could be interesting to provide them tools that bridge the gap between source code repositories, mailing lists, and bug trackers. Nowadays, some tools start to create such links between repositories. For instance, modern source forges allow their users to indicate which commit fixes the issue described in a given closed ticket. Bacchelli et al. [9] proposed Remail, a plugin for Eclipse
showing the e-mail discussions that relate to given lines of code. More generally, tools unifying and presenting the information from different repositories would offer a better support to the most active contributors. Tools that create links between different types of activity would be useful, too. For instance, a tool that supports co-evolution of code and documentation would be helpful for active contributors involved in both of these activity types.

Another envisaged tool is a dashboard that presents to the persons responsible of the evolution of the ecosystem (or the persons who want to manage its evolution) visualisations and metrics of social aspects of the considered software ecosystem. The dashboard should include identity merge algorithms and activity type detection algorithms for offering a better insight of the persons who actually contribute to the ecosystem. Our application framework offers such algorithms as well as initial configuration parameters. Even if these parameters should probably be adapted to the specificities of the considered ecosystem, they avoid a tedious implementation by the dashboard’s users. The dashboard could also offer support for projects belonging to an ecosystem in order to evaluate their fitness compared to other projects. The idea here is to highlight projects which are different to the other projects being parts of the ecosystem, the observed differences maying be an indicator of health problems for the project.

Our studies reveal the existence of sub-communities presenting different behaviours. A future dashboard should show these heterogeneities and should present potential issues in each of the discovered sub-communities. For instance, our studies confirm the presence of power laws in the workload of studied sub-communities. This may lead to a low bus factor, which is synonym to high risk for the ecosystem, or a part of this. This is particularly true in open source software ecosystems, in which most of contributors enter and leave the community at will. An interesting feature of our future dashboard is the ability to show most critical persons, i.e. contributors who must be retained in the ecosystem. The analysis of how critical a given contributor is should be based on his relative workload with relation to the sub-communities in which he is involved. Contributors involved in activity types (such as database) involving few persons should also be the object of a particular attention.

A possible extension to our empirical studies is the analysis of the migration patterns of contributors. We observed that some contributors leave a project belonging to GNOME to become active in an other project of this ecosystem. We would like to study the dynamic of contributors joining and leaving the ecosystem or its projects. In particular, we would determine if we can distinguish projects that tend to attract new contributors in the ecosystem from projects that tend to let the contributors leave the ecosystem. Similarly, we are interested in internal migrations. For instance, we would determine if some projects tend to attract contributors who are already involved in other projects belonging to the ecosystem.
This chapter summarises the contributions of this dissertation. It discusses the limitations of the methodologies followed for carrying out our empirical studies and for designing our framework for analysing the evolution of software ecosystems. The chapter concludes by presenting the open research perspectives of this thesis.
9.1 Contributions

Part I of this dissertation discussed the necessity to study the social aspects that surround a software ecosystem. In Section 1.1, we presented our research goals:

Goal 1: Extracting and post-processing historical data from open source ecosystems;

Goal 2: Empirical analysis and interpretation of the extracted data;

Goal 3: Understanding of social interactions in open source ecosystems.

In Part II we presented an application framework that offers the tools needed for reaching the first goal. We used this framework for extracting and analysing the data handled by repository tools used for managing software projects. The framework offers a unified process for the processing of information related to software ecosystem evolution. It combines both existing (such as CVSAnaly2) and custom-built tools. The latter are:

- In the framework post-processing layer, a tool for detecting and merging the repository accounts belonging to the same physical person. This tool is able to handle data from version control systems, mailing lists and bug tracker accounts. It provides several identity merge algorithms and can easily be extended to support other kinds of software repositories.

- In the framework post-processing layer, a tool for associating an activity to each file touched in a source code repository. The used association algorithm being based on an external rule file, the tool can be adapted to the particularities of the studied software system.

- In the analysis layer, a set of tools for analysing and for visually representing different aspects of software ecosystem communities.

Part III presented methodologies for reaching the second and the third research goals. We applied these methodologies for studying GNOME, a popular OSS ecosystem involving hundreds of projects and thousands of contributors. We carried out three empirical analyses by interpreting data extracted by the tools belonging to our software framework. The studies focused on social interactions in GNOME.
9.1. CONTRIBUTIONS

We studied the software communities that are involved in source code repositories, mailing lists and bug trackers of Evince, Brasero, Wine, and GNOME projects. By analysing their size, their activity, and the contributors who are involved in different communities, we conclude that the most active contributors are involved in the three considered types of repositories, for two of the studied software systems. For each studied community, we also found a strong evidence for the Pareto principle and power law behaviour. In particular, we found that more than half of the software projects belonging to GNOME respect the 20%-80% rule for their commit activity or have an even more unbalanced workload distribution. The study of workload imbalance reveals that, after an initial period, the four studied OSS projects tend to present a stable or slightly increasing workload imbalance. In this stabilisation phase, the workload distributions seem to suggest that most of the workload is carried out by a few number of contributors.

We considered in more detail the workload distribution by statistically studying the specialisation of contributors involved in source code repositories of the GNOME ecosystem. We defined fourteen activity types to which files belonging to GNOME projects can be associated. We also defined metrics for measuring the workload associated to each contributor, each GNOME project and each activity type. Finally, we studied how the workload varies across considered projects and contributors.

We confirmed that there is no such thing as a uniform ecosystem of projects and contributors: when taking into account the activity types and the workload, there is a lot of variation across projects and across contributors, but with a clear preference towards the activity types of coding, localisation, development documentation, and building.

We observed that project activity across the GNOME ecosystem follows a log-normal distribution. We also provided evidence that a project having a high number of activity types or having a large developer community is more active than a project having a small number of activity types or a small developer community. Among the considered activity types, coding, development documentation, localisation, and build are the four most important ones, at the ecosystem level as well as at the level of individual projects. It is also these four activity types that attract most of the authors involved in projects. However, while it is coding that concentrates most of a project’s workload, localisation and development documentation attract most of the contributors. We observed that most projects concentrate their workload in few activity types. We found evidence that highly specialised projects are also projects including few activity types. Moreover, as projects contain more contributors, they are more commonly translators rather than coders.

By trying to understand how workload varies across developers belonging to the same activity community, we observed that most authors are occasional contributors with little activity, while relatively few authors are frequent contributors that are very active. We
also investigated the factors that influence how active developers are and found that the more activity types a developer participates in, or the more projects he contributes to, the more active he is. Moreover, we observed that when contributing to different projects within GNOME, developers prefer to specialise in a small number of activity types. We provided evidence that coding, development documentation, localisation, and build are also the four activity types in which developers tend to specialise. While frequent developers principally code, occasional ones specialise in localisation. Both frequent as well as occasional developers are attracted to development documentation and build. Our empirical study revealed that most developers contributing to coding, development documentation, and localisation in one project, do so as well in other projects they contribute to within the GNOME ecosystem. However, as developers become involved in more projects, it is more common for them to participate in localisation rather than coding across the different projects. On the other hand, developers contributing to scarce activities (such as database) “wear many hats”, i.e they contribute to other activity types when their preferred ones are not available.

By studying the evolutionary aspect of the fourteen considered development activity types, we found that GNOME developers tend to restrict themselves to a limited number of activity types. Moreover, we observed a tendency of specialisation towards less activities. Nevertheless, the more active a developer is, the more he tends to spread his effort over different types of activity. During the entire time life of studied projects, the coding, and localisation activity types involve the most of contributors. We discovered that, during the last years, the workload of each activity community is decreasing.

Summing up, we gained understanding of how communities involved in an open source software ecosystem work and interact together over time by showing that

- Contributors use different user accounts for communicating, and finding them is a hard and time consuming task.
- The contributor community is actually made of sub-communities in which members work and collaborate in different ways.
- The contributors have different profiles: some of them use a single communication tool, while others communicate through different types of tools; some of them are specialised in an activity type, while others are generalists; some of them have an important and regular workload while others are only occasional contributors; some of them are involved in different projects while others only work on a single project. There exist correlations between these aspects of contributors involvement.
- The contributors tend to restrict the number activity types in which they are involved over time.
9.2. THREATS TO VALIDITY AND LIMITATIONS

- The different studied activity types coexist: none of them presents a decreasing activity for the benefit of another one.

- In the last years, the global workload of GNOME decreases.

9.2 Threats to Validity and Limitations

The methodology followed and the tools used in our empirical studies suffer from different threats to validity and limitations that are discussed in the dedicated chapters. In this section, we summarise those that we believe to be the more important and we present potential and partial remedies.

9.2.1 Identity merging

In each of our empirical studies, we used an identity merging tool for detecting and merging identities that belong to the same physical person. Because our empirical studies have been carried out during the 4 years this thesis required, our understanding of the relations between GNOME contributors become more accurate as we progressed in our PhD research. As the used merge algorithms are partially based on the specificities of the considered ecosystem, our ability to propose an accurate identity merge model improved over time. Despite our best effort and the already discussed measures we applied to reduce errors in the merging process, identity merge algorithms are not perfect, as explained in Section 4.7.

Our best hope to remediate to this limitation comes from the current evolution of tools used in software ecosystems. These tools tend to be combined in meta-tools, such as software forges, that centralise the identification and the authentication of contributors. Such tools are therefore able to show explicit links between the accounts used in a growing number of activities needed for evolving a software system. Nevertheless, there is still a privacy issue, already encountered during our empirical analyses, that may prevent us from accessing this information, even if it is technically available.
9.2.2 Activity type detection

The detection of activity types also suffers from imperfections. The fact that we decided to associate a single activity type to each considered file in the ecosystem repository can be seen as a limitation, since some files may be involved in several activity types. For instance, a .bmp file can be considered as an image file and can be simultaneously part of the user documentation. The association to an activity type is based on the path, the name, and the extension of the file. In other words, the association is based on, generally implicit, conventions. Even if we observed that these conventions are generally respected in the GNOME ecosystem, there is no strong guarantee they are. Even worse, certain file extensions are used in different contexts. For instance, the .tmp file extension commonly indicates a temporary file, i.e. a file that has been generated as an intermediate artefact and that can be safely removed.

However, several processes can generate such a file: program compilation, documentation generation, etc. The .GED extension file may refer to a genealogy data exchange file or to a compiled configuration file. Association rules can use other information, such as the path of the considered file or the project context, to disambiguate the activity type. An in-depth analysis of the file content may offer a more accurate classification by recognising type signature of a file, such as its magic number \[135\], a textual or numerical constant value placed at the beginning of a file and acting as a format indicator. Such an analysis on all the files belonging to the GNOME ecosystem is nevertheless extremely time consuming and out of the topic of this dissertation.

Another refinement could be the analysis of changes made to the files, in order to detect other types of activity such as bug fixing, comments adding, etc. The study of activity types related to other project repositories, such as e-mail discussions and bug reports handling, will certainly be an interesting further study. For instance, Bergstrom [13] studied a member of the Reddit community suspected to be a troll, an undesirable sort of contributor. Studer et al. [161] distinguished between regular, short term, and punctual mailing lists contributors.

9.2.3 Generalisation

This dissertation focused on the social aspects of the community of contributors that surrounds an OSS ecosystem. In order to reach our research goals and to answer the related questions, we studied GNOME. However, a general threat to validity for our studies is the generalisability of our study of this ecosystem. GNOME is an important
ecosystem, which contains a lot of various projects. We therefore estimate that the risk of sensitivity to specificities is limited. There may nevertheless be ecosystem specificities due to its global policies. For instance, if we consider the OpenBSD ecosystem, the developers strongly insist on the software security and the documentation quality aspects. In order to assess the generalisability of the obtained results, a replication study based on another OSS ecosystem (such as KDE) should be considered. The processes suggested or required in order to organise the studied ecosystem evolution may also affect the results of our empirical studies. We did not find evidence for such formal processes in GNOME, except throughout translation tools like Damned Lies which helps the GNOME translator to share the translation workload. A replication study is facilitated by the adaptability of the application framework used in our empirical studies.
Part V

Appendices
List of insignificant words for the identity merge algorithms

Table A.1 contains a list of common words that are removed from identity labels (logins, names and e-mail addresses) during the normalisation process, as they do not offer interesting information to merge identities and give rise to false positives. This list has been obtained on the basis of common words in OSS development and a manual inspection of the analysed projects. Project-specific terms are shown in the fourth column. When applying a merge algorithm to yet another project, this list needs to be extended further.
Table A.1: Insignificant words occurring in identity labels

<table>
<thead>
<tr>
<th>prefixes</th>
<th>technical terms</th>
<th>other terms</th>
<th>project-specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>mr.</td>
<td>administrator</td>
<td>spam</td>
<td>evince</td>
</tr>
<tr>
<td>mrs.</td>
<td>admin.</td>
<td>bug</td>
<td>brasero</td>
</tr>
<tr>
<td>miss</td>
<td>support</td>
<td>bugs</td>
<td>bugzilla</td>
</tr>
<tr>
<td>ms.</td>
<td>development</td>
<td>root</td>
<td>gnome</td>
</tr>
<tr>
<td>prof.</td>
<td>dev.</td>
<td>mailing</td>
<td>linux</td>
</tr>
<tr>
<td>pr.</td>
<td>developer</td>
<td>list</td>
<td></td>
</tr>
<tr>
<td>dr.</td>
<td>maint.</td>
<td>contact</td>
<td></td>
</tr>
<tr>
<td>ir.</td>
<td>maintainer</td>
<td>project</td>
<td></td>
</tr>
<tr>
<td>rev.</td>
<td>i18n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ing.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jr.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d.d.s.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ph.d.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>capt.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lt.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Activity type rules

Rules used to assign each file to an activity type. The rule for each activity type is defined by a regular expression. If the expression matches the file’s path, the activity type is associated to the file. The rules are assessed in sequence. Among the rules matching the file, the last one is used to determine the activity type. We do not allow for multiple classification, as this poses problems with the definition of some metrics and the statistical analysis of some results.

<table>
<thead>
<tr>
<th>Activity type acronym</th>
<th>Regular expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>.*</td>
</tr>
<tr>
<td>unknown</td>
<td>./((s</td>
</tr>
<tr>
<td></td>
<td>./contributors</td>
</tr>
<tr>
<td></td>
<td>./page</td>
</tr>
<tr>
<td></td>
<td>./1</td>
</tr>
<tr>
<td></td>
<td>./zabw</td>
</tr>
<tr>
<td></td>
<td>*/install</td>
</tr>
<tr>
<td></td>
<td>./chm</td>
</tr>
<tr>
<td></td>
<td>*/copyright</td>
</tr>
<tr>
<td></td>
<td>./css</td>
</tr>
<tr>
<td></td>
<td>*/plan</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Activity type acronym</th>
<th>Regular expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*/txt((/\bak)?)</td>
<td>.*/credits</td>
</tr>
<tr>
<td>*/notes</td>
<td>*/licence</td>
</tr>
<tr>
<td>.*/txt((/\old)?)</td>
<td>*/.man</td>
</tr>
<tr>
<td>*/howto</td>
<td>*/.docx</td>
</tr>
<tr>
<td>*/.rtf</td>
<td>*/.ics</td>
</tr>
<tr>
<td>*/faq</td>
<td>*/maintainers</td>
</tr>
<tr>
<td>*/.tex</td>
<td>*/documenters</td>
</tr>
<tr>
<td>*/copying</td>
<td>*/copying</td>
</tr>
<tr>
<td>*/.sgml</td>
<td><em>/copying.</em></td>
</tr>
<tr>
<td>*/.committers</td>
<td>*/.pdf</td>
</tr>
<tr>
<td>*/.vcf</td>
<td><em>/doc(s?)/</em>.</td>
</tr>
<tr>
<td>*/.thanks</td>
<td>*/.xsd</td>
</tr>
<tr>
<td>*/.schemas</td>
<td><em>/help(s?)/</em>.</td>
</tr>
<tr>
<td>*/.authors</td>
<td>*/.texi</td>
</tr>
<tr>
<td>*/.doc</td>
<td>*/.bugs</td>
</tr>
<tr>
<td>*/.documenters</td>
<td>*/.gnnumeric</td>
</tr>
<tr>
<td>.*/.png</td>
<td>.*/.ppm</td>
</tr>
<tr>
<td>*/.icns</td>
<td>*/.eps</td>
</tr>
<tr>
<td>*/.pgm</td>
<td>*/.jpg</td>
</tr>
<tr>
<td>*/.chm</td>
<td>*/.xbm</td>
</tr>
<tr>
<td>*/.jpeg</td>
<td>*/.bmp</td>
</tr>
<tr>
<td>*/.chm</td>
<td>*/.vdx</td>
</tr>
<tr>
<td>*/.gif</td>
<td>*/.sgv(z?)</td>
</tr>
<tr>
<td>*/.nsh</td>
<td>*/.ico</td>
</tr>
<tr>
<td>*/.xcf</td>
<td></td>
</tr>
<tr>
<td>.*/.potfiles.in(~?)</td>
<td>.*/.i18ns(~?)</td>
</tr>
<tr>
<td>.*/.pot(~?)</td>
<td>/po/.*</td>
</tr>
<tr>
<td>/strings.properties</td>
<td>.*/.mo(~?)</td>
</tr>
<tr>
<td>*/.linguas</td>
<td>*/.wxl</td>
</tr>
<tr>
<td>*/.gmo(~?)</td>
<td>*/.resx(~?)</td>
</tr>
<tr>
<td><em>/locale(s?)/</em>.</td>
<td>*/.po(~?)</td>
</tr>
<tr>
<td>.*.charset(~?)</td>
<td></td>
</tr>
<tr>
<td>.*/.glade(d?)((/\bak)?)(~?)</td>
<td>.*/.desktop</td>
</tr>
<tr>
<td>.*.xul(~?)</td>
<td>*/.ui</td>
</tr>
<tr>
<td>.*/gladed(d?)((/\bak)?)(~?)</td>
<td>*/.xpm</td>
</tr>
<tr>
<td>.*.gladep(d?)((/\bak)?)(~?)</td>
<td>*/.theme</td>
</tr>
<tr>
<td>.*/mp3</td>
<td>.*/mp4</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Activity type acronym</th>
<th>Regular expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>media</td>
<td><em>/media(s?)/.</em></td>
</tr>
<tr>
<td></td>
<td>∗..pfm</td>
</tr>
<tr>
<td></td>
<td>∗..mpv</td>
</tr>
<tr>
<td></td>
<td>∗..mml</td>
</tr>
<tr>
<td></td>
<td><em>/font(s?)/.</em></td>
</tr>
<tr>
<td></td>
<td>∗..gnc</td>
</tr>
<tr>
<td></td>
<td>∗..ogv</td>
</tr>
<tr>
<td></td>
<td>∗..mml</td>
</tr>
<tr>
<td></td>
<td><em>/icon(s?)/.</em></td>
</tr>
<tr>
<td></td>
<td>∗..shape</td>
</tr>
<tr>
<td></td>
<td>∗..wav</td>
</tr>
<tr>
<td></td>
<td>∗..au</td>
</tr>
<tr>
<td></td>
<td>*/.otf(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..gnc</td>
</tr>
<tr>
<td></td>
<td>∗..mov</td>
</tr>
<tr>
<td></td>
<td>∗..mml</td>
</tr>
<tr>
<td></td>
<td>*/.sfd(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..pgn</td>
</tr>
<tr>
<td></td>
<td>∗..mid</td>
</tr>
<tr>
<td></td>
<td>∗..xspf</td>
</tr>
<tr>
<td></td>
<td>*/.ttf(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..cdf</td>
</tr>
<tr>
<td></td>
<td>*/.m4f</td>
</tr>
<tr>
<td></td>
<td>∗..ps</td>
</tr>
<tr>
<td></td>
<td>*/.afm</td>
</tr>
<tr>
<td></td>
<td>∗..bse</td>
</tr>
<tr>
<td></td>
<td>*/.pls</td>
</tr>
<tr>
<td></td>
<td>∗..omf</td>
</tr>
<tr>
<td></td>
<td>*/.pfb</td>
</tr>
<tr>
<td></td>
<td>∗..cur</td>
</tr>
<tr>
<td>Coding code</td>
<td>*/.dmg(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..swg(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.so(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..i(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.h((pp)?)((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..exe(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.oafinfo(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..pyd(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.awk(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..scm(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.glsl(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..patch(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.c(((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..script(s?)/.*</td>
</tr>
<tr>
<td></td>
<td>*/.asp(x?)((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..src/*</td>
</tr>
<tr>
<td></td>
<td>*/.m((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..cs(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.h.win32((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..s(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.r((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..cxx(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.h\ .template((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..asm(x?)(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.py((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..y((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.gi((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..t((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.php((\ swp)?)(d?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..pyc(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.js((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..rb((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.c\ .template((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..hg((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.pm((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..dll(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.cc((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..sh((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.php(((\ swp)?)(d?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..el((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.hh((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..o(~?)</td>
</tr>
<tr>
<td></td>
<td>*/.xs((\ swp)?)(~?)</td>
</tr>
<tr>
<td></td>
<td>∗..pl((\ swp)?)(~?)</td>
</tr>
</tbody>
</table>

Continued on next page
Table B.1 – continued from previous page

<table>
<thead>
<tr>
<th>Activity type acronym</th>
<th>Regular expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*/.h.tmpl((.swp)?)(~?)</td>
<td>.*/.mm((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.idl((.swp)?)(~?)</td>
<td>.*/.idl(~?)</td>
</tr>
<tr>
<td>.*/.xpt((.swp)?)(~?)</td>
<td>.*/.ccg((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.c.tmpl((.swp)?)(~?)</td>
<td>.*/.snk((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.inc((.swp)?)(~?)</td>
<td>.*/.jar(~?)</td>
</tr>
<tr>
<td>.*/.cpp((.swp)?)(~?)</td>
<td>.*/.gob((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.vapi((.swp)?)(~?)</td>
<td>.*/.giv((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.dtd((.swp)?)(~?)</td>
<td>.*/.gidl((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.giv((.swp)?)(~?)</td>
<td>.*/.ada((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.defs((.swp)?)(~?)</td>
<td>.*/.tcl((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.vbs((.swp)?)(~?)</td>
<td>.*/.java((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.nib((.swp)?)(~?)</td>
<td>.*/.sed((.swp)?)(~?)</td>
</tr>
<tr>
<td>.*/.vala((.swp)?)(~?)</td>
<td>.<em>/.svn(.</em>)</td>
</tr>
<tr>
<td>.*/.doap</td>
<td>.*/.sln</td>
</tr>
<tr>
<td>.*/.mdp</td>
<td>.<em>/.config(s?).</em>/.*/.jp</td>
</tr>
<tr>
<td>.*/.anjuta</td>
<td>.*/.dsw</td>
</tr>
<tr>
<td>.*/.gnorba</td>
<td>.*/.project</td>
</tr>
<tr>
<td>.*/.pgp(~?)</td>
<td>.*/.ini</td>
</tr>
<tr>
<td>.*/.prefs</td>
<td>.*/.vsprops</td>
</tr>
<tr>
<td>.*/.gpg(~?)</td>
<td>.*/.config</td>
</tr>
<tr>
<td>.*/.vmrc</td>
<td>.*/.cproj</td>
</tr>
<tr>
<td>.*/.gpg.pub(~?)</td>
<td>.*/.xml</td>
</tr>
<tr>
<td>.*/.cproj</td>
<td>.*/.cbproj</td>
</tr>
<tr>
<td>.*/.pgp.pub(~?)</td>
<td>.*/.dsp</td>
</tr>
<tr>
<td>.*/.emacs</td>
<td>.*/.groupproj</td>
</tr>
<tr>
<td>.*/.xcconfig</td>
<td>.*/.plist</td>
</tr>
<tr>
<td>.*/.pbxproj</td>
<td>.*/.anjuta\session</td>
</tr>
<tr>
<td>.<em>/.<em>setting(s?).</em>/.</em>/.jp</td>
<td>.*/.conf</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Activity type acronym</th>
<th>Regular expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*.deps</td>
<td>.*.wxiproj</td>
</tr>
<tr>
<td>.*.am(~?)</td>
<td>.*.mp4</td>
</tr>
<tr>
<td>.*.builder</td>
<td>.*.lo</td>
</tr>
<tr>
<td>.*.target</td>
<td>.*.iss</td>
</tr>
<tr>
<td>.*.vcproj((/.filters((in)?))?)</td>
<td>.*.wxi</td>
</tr>
<tr>
<td>.*/configure((..+))</td>
<td>.*.wxs</td>
</tr>
<tr>
<td>.*/mkbundle..+</td>
<td>.*.in</td>
</tr>
<tr>
<td>.*/autogen((.+?)sh</td>
<td>.*.wpj</td>
</tr>
<tr>
<td>.*/vc(x?)proj(i?)n((. filters ((in)??))?)</td>
<td>.\nsi</td>
</tr>
<tr>
<td>.<em>/changelog.</em></td>
<td>.<em>readme.</em></td>
</tr>
<tr>
<td>.*/dia(~?)</td>
<td>.*.ical</td>
</tr>
<tr>
<td>.*/changes</td>
<td>.*status</td>
</tr>
<tr>
<td>.*/fixme</td>
<td>.*\doxi</td>
</tr>
<tr>
<td>.<em>/todo.</em></td>
<td>.<em>hacking.</em></td>
</tr>
<tr>
<td>.<em>/news.</em></td>
<td>.*roadmap</td>
</tr>
<tr>
<td>.*/rst</td>
<td>.<em>//devel(-?)doc(s)?/.</em></td>
</tr>
<tr>
<td>.*/sql</td>
<td>.*\sqlite</td>
</tr>
<tr>
<td>.*/mdb</td>
<td>.*\yaml</td>
</tr>
<tr>
<td>.*/sdb</td>
<td>.*\dat</td>
</tr>
<tr>
<td>.*/yaml</td>
<td>.*\json</td>
</tr>
<tr>
<td>.*/db</td>
<td><em>/berkeleydb.</em>/.*</td>
</tr>
<tr>
<td>.<em>/test(s?)/.</em></td>
<td><em>/.test..</em></td>
</tr>
<tr>
<td>.<em>/test.</em>..*</td>
<td><em>/.test..</em></td>
</tr>
<tr>
<td>.<em>/library/.</em></td>
<td><em>/libraries/.</em></td>
</tr>
</tbody>
</table>

Table B.1: Activity type rules
Table C.1 presents some relevant excerpts taken from the GNOME mailing list archives found at [https://mail.gnome.org/archives](https://mail.gnome.org/archives).

<table>
<thead>
<tr>
<th>N.</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The number of hackers who are paid full-time to work on GNOME has grown from 5 to more than 50, working with hundreds of volunteer hackers.</td>
</tr>
<tr>
<td>2</td>
<td>GNOME Community Celebrates 10 Years of Software Freedom, Innovation and Industry Adoption: Since 1997, the GNOME project has grown from a handful of developers to a contributor base of coders, documenters, translators, interface designers, accessibility specialists, artists and testers numbering in the thousands.</td>
</tr>
<tr>
<td>3</td>
<td>Just on this note, let me state that I in no way consider translators as second-class citizens; nor documenters, UI dudes, general organisers, or anyone whatsoever just because they do not code.</td>
</tr>
</tbody>
</table>

Continued on next page
Table C.1 – continued from previous page

<table>
<thead>
<tr>
<th>N.</th>
<th>Message</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><em>The problem, of course, is that gnome <em>is</em> a code-centric organization. The coders are our sine qua non—without them, we have nothing. Translators are probably a close second to that—without them, we have no international coders. Past that, no group of people in the project are indispensable to the current state of the project, or even close to it.</em></td>
<td><a href="https://mail.gnome.org/archives/marketing-list/2007-February/msg00027.html">https://mail.gnome.org/archives/marketing-list/2007-February/msg00027.html</a></td>
</tr>
<tr>
<td>5</td>
<td><em>Furthermore, in GNOME, we have many translators that started out with just translating, but continued with fixing bugs, and some are full time coders now. We should be proud of this integration in the community.</em></td>
<td><a href="https://mail.gnome.org/archives/foundation-list/2007-September/msg00050.html">https://mail.gnome.org/archives/foundation-list/2007-September/msg00050.html</a></td>
</tr>
<tr>
<td>6</td>
<td><em>As you have pointed out yourself, translators are usually not hackers/coders.</em></td>
<td><a href="https://mail.gnome.org/archives/gnome-web-list/2001-August/msg00073.html">https://mail.gnome.org/archives/gnome-web-list/2001-August/msg00073.html</a></td>
</tr>
</tbody>
</table>

Table C.1: GNOME mailing list discussions
This appendix summarises all metrics used in Chapter 7, as well as some other similar ones. Table D.1 lists all workload metrics, table D.2 lists all involvement metrics.

Based on the metrics from Table D.1 and Table D.2, relative workload and relative involvement metrics can be defined that are listed in Table D.3.

Finally, Table D.4 lists all relevant specialisation metrics, i.e. aggregated metrics where we compute the Gini index over some other metric, in order to determine if there is a type specialisation effect, i.e. an imbalance or unequal spread in the distribution over the activity types.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>APTI(p,a,t)</td>
<td>Author-Project-Type Involvement</td>
<td>indicates whether determines author a has been involved in activity type t for project p</td>
<td>1 if $APTW(p,a,t) &gt; 0$, 0 otherwise</td>
</tr>
<tr>
<td>ATI(a,t)</td>
<td>Author-Type Involvement</td>
<td></td>
<td>$\sum_{p_i \in P} APTI(p_i,a,t)$</td>
</tr>
<tr>
<td>PTI(p,t)</td>
<td>Project-Type Involvement</td>
<td></td>
<td>$\sum_{a_j \in A} APTI(p,a_j,t)$</td>
</tr>
<tr>
<td>PAI(p,a)</td>
<td>Project-Author Involvement</td>
<td></td>
<td>$\sum_{t_k \in T} APTI(p,a,t_k)$</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPA(a)</td>
<td>Number of Projects for an Author.</td>
<td>Number of projects an author is involved in.</td>
<td>( \sum_{p_i \in P} \left( \max_{t_k \in T} (APT I(p_i, a, t_k)) \right) )</td>
</tr>
<tr>
<td>NTA(a)</td>
<td>Number of Types for an Author.</td>
<td>Number of activity types an author is involved in.</td>
<td>( \sum_{t_k \in T} \left( \max_{p_i \in P} (APT I(p_i, a, t_k)) \right) )</td>
</tr>
<tr>
<td>NAT(t)</td>
<td>Number of Authors for a Type.</td>
<td>Number of authors involved in a type of activity.</td>
<td>( \sum_{a_j \in A} \left( \max_{p_i \in P} (APT I(p_i, a_j, t)) \right) )</td>
</tr>
<tr>
<td>NPT(t)</td>
<td>Number of Projects for a Type.</td>
<td></td>
<td>( \sum_{p_i \in P} \left( \max_{a_j \in A} (APT I(p_i, a_j, t)) \right) )</td>
</tr>
<tr>
<td>NAP(p)</td>
<td>Number of Authors for a Project.</td>
<td></td>
<td>( \sum_{a_j \in A} \left( \max_{t_k \in T} (APT I(p, a_j, t_k)) \right) )</td>
</tr>
<tr>
<td>NTP(p)</td>
<td>Number of Types for a Project.</td>
<td></td>
<td>( \sum_{t_k \in T} \left( \max_{a_j \in A} (APT I(p, a_j, t_k)) \right) )</td>
</tr>
</tbody>
</table>

Table D.2: Definition of absolute involvement metrics
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>APTW(p,a,t)</td>
<td>Author-Project-Type Workload</td>
<td>Number of touches of files of activity type, ( t ) by author ( a ) for a project ( p ) over its entire history.</td>
<td>extracted from source code repository</td>
</tr>
<tr>
<td>ATW(a,t)</td>
<td>Author-Type Workload</td>
<td>( \sum_{p_i \in P} \sum_{t_k \in T} APTW(p_i, a, t_k) )</td>
<td>( \sum_{a_j \in A} \sum_{t_k \in T} APTW(p, a_j, t_k) )</td>
</tr>
<tr>
<td>PTW(p,t)</td>
<td>Project-Type Workload</td>
<td>( \sum_{a_j \in A} \sum_{t_k \in T} APTW(p, a_j, t_k) )</td>
<td>( \sum_{p_i \in P} \sum_{t_k \in T} APTW(p_i, a, t_k) )</td>
</tr>
<tr>
<td>PAW(p,a)</td>
<td>Project-Author Workload</td>
<td>( \sum_{p_i \in P} \sum_{t_k \in T} APTW(p_i, a, t_k) )</td>
<td>( \sum_{a_j \in A} \sum_{t_k \in T} APTW(p, a_j, t_k) )</td>
</tr>
<tr>
<td>AW(a)</td>
<td>Author Workload</td>
<td>( \sum_{p_i \in P} \sum_{t_k \in T} APTW(p_i, a, t_k) )</td>
<td>( \sum_{a_j \in A} \sum_{t_k \in T} APTW(p, a_j, t_k) )</td>
</tr>
<tr>
<td>PW(p)</td>
<td>Project Workload</td>
<td>( \sum_{a_j \in A} \sum_{p_j \in P} \sum_{t_k \in T} APTW(a_i, p_j, t_k) )</td>
<td>( \sum_{a_j \in A} \sum_{p_j \in P} \sum_{t_k \in T} APTW(a_i, p_j, t_k) )</td>
</tr>
<tr>
<td>TW(t)</td>
<td>Type Workload</td>
<td>( \sum_{a_j \in A} \sum_{p_j \in P} \sum_{t_k \in T} APTW(a_i, p_j, t_k) )</td>
<td>( \sum_{a_j \in A} \sum_{p_j \in P} \sum_{t_k \in T} APTW(a_i, p_j, t_k) )</td>
</tr>
<tr>
<td>GW</td>
<td>Global Workload</td>
<td>( \sum_{a_j \in A} \sum_{p_j \in P} \sum_{t_k \in T} APTW(a_i, p_j, t_k) )</td>
<td>( \sum_{a_j \in A} \sum_{p_j \in P} \sum_{t_k \in T} APTW(a_i, p_j, t_k) )</td>
</tr>
</tbody>
</table>

Table D.1: Definitions of absolute workload metrics

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPW(p,a)</td>
<td>Relative Author-Project Workload</td>
<td>workload of a given author in a particular project, relative to the author workload.</td>
<td>( \frac{PAW(p,a)}{AW(a)} )</td>
</tr>
<tr>
<td>RATW(a,t)</td>
<td>Relative Author-Type Workload</td>
<td>workload of a given author for a particular type, relative to the author workload.</td>
<td>( \frac{ATW(a,t)}{AW(a)} )</td>
</tr>
<tr>
<td>RPAW(p,a)</td>
<td>Relative Project-Author Workload</td>
<td>workload of a particular author in a given project, relative to the project workload.</td>
<td>( \frac{PAW(p,a)}{PW(p)} )</td>
</tr>
<tr>
<td>RPTW(p,t)</td>
<td>Relative Project-Type Workload</td>
<td>workload in a given project for a particular type, relative to the project workload.</td>
<td>( \frac{PTW(p,t)}{PW(p)} )</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RTAW(a,t)$</td>
<td>Relative Type-Author Workload</td>
<td>workload of a particular author for a given type, relative to the type workload.</td>
<td>$\frac{ATW(a,t)}{TW(t)}$</td>
</tr>
<tr>
<td>$RTPW(p,t)$</td>
<td>Relative Type-Project Workload</td>
<td>workload in a particular project for a given type, relative to the type workload.</td>
<td>$\frac{PTW(p,t)}{TW(t)}$</td>
</tr>
<tr>
<td>$RAW(a)$</td>
<td>Relative Author Workload</td>
<td>workload of a particular author, relative to the global workload.</td>
<td>$\frac{AW(a)}{GW}$</td>
</tr>
<tr>
<td>$RPW(p)$</td>
<td>Relative Project Workload</td>
<td>workload of a particular project, relative to the global ecosystem workload.</td>
<td>$\frac{PW(p)}{GW}$</td>
</tr>
<tr>
<td>$RTW(t)$</td>
<td>Relative Type Workload</td>
<td>workload of a particular type, relative to the global workload.</td>
<td>$\frac{TW(t)}{GW}$</td>
</tr>
<tr>
<td>$RAPI(p,a)$</td>
<td>Relative Author-Project Involvement</td>
<td>Involvement of a given author in a particular project, relative to the number of types the author is involved in.</td>
<td>$\frac{PAI(p,a)}{NPA(a)}$</td>
</tr>
<tr>
<td>$RATI(a,t)$</td>
<td>Relative Author-Type Involvement</td>
<td>Involvement of a given author for a particular type, relative to the number of projects in which the author is involved.</td>
<td>$\frac{ATI(a,t)}{NPA(a)}$</td>
</tr>
<tr>
<td>$RPAI(p,a)$</td>
<td>Relative Project-Author Involvement</td>
<td>Involvement of a particular author in a given project, relative to the number of types in the project</td>
<td>$\frac{PAI(p,a)}{NTP(p)}$</td>
</tr>
<tr>
<td>$RPTI(p,t)$</td>
<td>Relative Project-Type Involvement</td>
<td>Involvement in a given project for a particular type, relative to the number of authors involved in the project.</td>
<td>$\frac{PTI(p,t)}{NAP(p)}$</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RTAI(a,t)$</td>
<td>Relative Type-Author Involvement</td>
<td>Involvement of a particular author for a given type, relative to the number of projects having this type.</td>
<td>$\frac{ATI(a,t)}{NPT(t)}$</td>
</tr>
<tr>
<td>$RTPI(p,t)$</td>
<td>Relative Type-Project Involvement</td>
<td>Involvement in a particular project for a given type, relative to the number of authors working this type.</td>
<td>$\frac{PTI(p,t)}{NAT(t)}$</td>
</tr>
</tbody>
</table>

Table D.3: Definition of relative metrics
## APPENDIX D. WORKLOAD AND INVOLVEMENT METRICS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full name</th>
<th>Description</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS(a)</td>
<td>Author Workload Specialisation</td>
<td>Imbalance of workload across activity types in which author a is contributing, over all projects.</td>
<td>$Gini_{i_k \in T}(ATW(a, t_k))$</td>
</tr>
<tr>
<td>$PWS(p)$</td>
<td>Project Workload Specialisation</td>
<td>Imbalance of workload across activity types for a project p, over all authors contributing to this project.</td>
<td>$Gini_{i_k \in T}(PTW(p, t_k))$</td>
</tr>
<tr>
<td>AIS(a)</td>
<td>Author Involvement Specialisation</td>
<td>Imbalance of involvement across activity types in which author a is contributing, over all projects.</td>
<td>$Gini_{i_k \in T}(ATI(a, t_k))$</td>
</tr>
<tr>
<td>$PIS(p)$</td>
<td>Project Involvement Specialisation</td>
<td>Imbalance of involvement across activity types for a project p, over all authors contributing to this project.</td>
<td>$Gini_{i_k \in T}(PTI(p, t_k))$</td>
</tr>
<tr>
<td>RAWS(a)</td>
<td>Relative Author Workload Specialisation</td>
<td>Imbalance of relative workload across activity types in which author a is contributing, over all projects.</td>
<td>$Gini_{i_k \in T}(RATW(a, t_k))$</td>
</tr>
<tr>
<td>RPWS(p)</td>
<td>Relative Project Workload Specialisation</td>
<td>Imbalance of relative workload across activity types for a project p, over all authors contributing to this project.</td>
<td>$Gini_{i_k \in T}(RPTW(p, t_k))$</td>
</tr>
<tr>
<td>RAIS(a)</td>
<td>Relative Author Involvement Specialisation</td>
<td>Imbalance of relative involvement across activity types in which author a is contributing, over all projects.</td>
<td>$Gini_{i_k \in T}(RATI(a, t_k))$</td>
</tr>
<tr>
<td>RPIS(p)</td>
<td>Relative Project Involvement Specialisation</td>
<td>Imbalance of relative involvement across activity types for a project p, over all authors contributing to this project.</td>
<td>$Gini_{i_k \in T}(RPTI(p, t_k))$</td>
</tr>
</tbody>
</table>

Table D.4: Definition of specialisation metrics
List of Figures

1.1 “In-the-small” view on software ecosystems ........................................... 10
1.2 Evolution of basic GNOME metrics............................................................... 15
2.1 General structure of an OSS community [121] ............................................. 20
3.1 Framework for extracting, processing and analysing open source software
ecosystems ........................................................................................................ 29
3.2 Enhanced Entity-relationship model [50] of the database for source code
repositories of software ecosystems ................................................................. 33
3.3 Examples of visualisations available in the framework ................................... 35
4.1 Example of contributors and their identities and associated labels in different
repositories. This merge model represents the ideal case in which an
identity merge algorithm would be able to correctly associate each identity
to its corresponding contributor ....................................................................... 41
4.2 Precision and recall for each parameter of identity detection algorithms ...... 58
4.3 Variation of precision and recall for the Brasero project ............................... 59
4.4 Variation of precision and recall for the Evince project ............................... 60
4.5 Variation of precision and recall for the Subversion project .......................... 61
## List of Figures

6.1 Visual representation of the Gini index ........................................ 85
6.2 Cumulative views of distribution of activity for Brasero, Evince, Wine, and GNOME. .................................................. 89
6.3 Number of projects belonging to the GNOME ecosystem that respect a given distribution of commit workload. ..................... 92
6.4 Overlaps of the top contributors in Brasero, Evince, and Wine ........ 93
6.5 Comparison of Gini, Theil, and Hoover indices for commit activity .... 95
6.6 Evolution of workload distributions for Brasero, Evince, and Wine .... 97

7.1 Boxplots of characteristics across GNOME projects ....................... 107
7.2 Identity matching steps. ............................................................ 108
7.3 Workload metrics ...................................................................... 111
7.4 Involvement metrics ................................................................... 112
7.5 The workload $PW(p)$ is distributed log-normally. ......................... 120
7.6 Linear relation between $NTP(p)$ and $\log PW(p)$ ......................... 121
7.7 Type workload .......................................................................... 123
7.8 Relative project workload specialisation .................................... 123
7.9 Relative project workload per activity type. ................................ 126
7.10 Boxplots for relative project involvement per activity type .......... 128
7.11 Correlations between NAP, PTW, and PTI. ................................. 131
7.12 Distribution of author workload. ................................................. 134
7.13 Correlation between $NTA$ and $AW$ .......................................... 135
7.14 Relative author-workload specialisation $RAWS(a)$. ...................... 137
7.15 Boxplots for $RATW(a,t)$ per activity type $t$ .............................. 139
7.16 Visualisations of occasional and frequent contributors. ............... 141
7.17 Boxplots for $RATI(a,t)$ per activity type $t$ ............................... 143
7.18 Correlation between $NPA$, $ATW$, and $ATI$. ......................... 148

8.1 Evolution of the yearly absolute workload per activity community .... 163
8.2 Evolution of the yearly relative workload per activity community ..... 163
List of Tables

1.1 Our publications related to the chapters of this dissertation. .................. 14

4.1 The four possibilities to determine whether a given pair of identities correctly belongs to the same identity merge. ....................... 46

4.2 Main characteristics of selected software projects. The reported values are those of the last version of each project. Values for May 2011. ............. 56

5.1 List of all activity types detected by the framework, by analysing the files committed in the version control repository of a software project. ........ 72

5.2 Excerpt of rules \((t, e)\) that may be used to identify the activity types from the file paths and file names. ................................. 74

6.1 Main characteristics of selected OSS projects. The reported values have been computed for the last extracted version before submission of [62], November 2010. ......................................................... 88

6.2 Best parameters for functions \(y = Cx^{-\alpha}\) fitting activity workloads. The fitted data is normalised and ordered by decreasing workload. ........ 91

7.1 Variation of characteristics across GNOME projects ...................... 106

7.2 Definitions of \(APTW\)-based project-level workload metrics. .......... 113

195
7.3 Definitions of APTI-based project-level involvement metrics. . . . . . . . . 114
7.4 Illustration of $\tilde{T}$ procedure and $\tilde{T}$-graph . . . . . . . . . . . . . 118
A.1 Insignificant words occurring in identity labels . . . . . . . . . . . . . . . . . 178
B.1 Activity type rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 183
C.1 GNOME mailing list discussions . . . . . . . . . . . . . . . . . . . . . . . . . 186
D.2 Definition of absolute involvement metrics . . . . . . . . . . . . . . . . . . 188
D.1 Definitions of absolute workload metrics . . . . . . . . . . . . . . . . . . . 189
D.3 Definition of relative metrics . . . . . . . . . . . . . . . . . . . . . . . . . . 191
D.4 Definition of specialisation metrics . . . . . . . . . . . . . . . . . . . . . . . 192


*Cited in page 117*

*Cited in page 119*

*Cited in page 117*

*Cited in page 18*

*2 citations in pages 20 and 151*

*Cited in page 43*

*Cited in page 18*

*Cited in page 18*

*Cited in page 164*

*Cited in page 164*
Cited in page 7

2 citations in pages 22 and 152

Cited in page 34

Cited in page 43

3 citations in pages 46, 108, and 109

Cited in page 108

6 citations in pages 81, 87, 115, 116, 120, and 134

Cited in page 81

Cited in page 162

Cited in page 80


Cited in page 18


2 citations in pages 14 and 27


5 citations in pages 14, 79, 87, 88, and 195


6 citations in pages 3, 9, 14, 17, 27, and 69


2 citations in pages 14 and 37


Cited in page 7


Cited in page 47


2 citations in pages 153 and 162


Cited in page 87


2 citations in pages 81 and 87


Cited in page 116


Cited in page 8


Cited in page 150


Cited in page 43


Cited in page 20


Cited in page 160


Cited in page 114


2 citations in pages 116 and 119


Cited in page 108
<table>
<thead>
<tr>
<th>Metric/Software</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate metric</td>
<td>83</td>
</tr>
<tr>
<td>Bicho</td>
<td>32, 42</td>
</tr>
<tr>
<td>Brasero</td>
<td>55, 87</td>
</tr>
<tr>
<td>CLOC</td>
<td>160</td>
</tr>
<tr>
<td>Complicity</td>
<td>23</td>
</tr>
<tr>
<td>CVSANaly2</td>
<td>31, 42, 106, 168</td>
</tr>
<tr>
<td>Damned Lies</td>
<td>9, 147, 151, 162, 173</td>
</tr>
<tr>
<td>Evince</td>
<td>55, 87</td>
</tr>
<tr>
<td>FLOSSMetrics</td>
<td>31</td>
</tr>
<tr>
<td>free software</td>
<td>7</td>
</tr>
<tr>
<td>Gini index</td>
<td>83, 111, 113</td>
</tr>
<tr>
<td>Git</td>
<td>31, 106</td>
</tr>
<tr>
<td>Goal-Question-Metric</td>
<td>80, 82</td>
</tr>
<tr>
<td>KDE</td>
<td>8</td>
</tr>
<tr>
<td>Levenshtein distance</td>
<td>49, 108</td>
</tr>
<tr>
<td>MLStats</td>
<td>32, 42</td>
</tr>
<tr>
<td>Onion model</td>
<td>19, 70, 98, 152</td>
</tr>
<tr>
<td>open source</td>
<td>7</td>
</tr>
<tr>
<td>OSS</td>
<td>7</td>
</tr>
<tr>
<td>R project</td>
<td>27, 86, 90, 114</td>
</tr>
<tr>
<td>SLOCCount</td>
<td>160</td>
</tr>
<tr>
<td>Small Project Observatory</td>
<td>22</td>
</tr>
</tbody>
</table>