Reducing Knowledge Loss in Open Source Software Projects

Mehvish Rashid, B.Sc., M.Sc.

A Dissertation submitted in fulfilment of the requirements for the award of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University

School of Computing

Supervisors: Dr. Paul M. Clarke, Prof. Rory V. O'Connor, Dr. Murat Yilmaz

March 2020

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ph.D is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____ ID No.: 17211694

(Mehvish Rashid)

Date: _____

Acknowledgements

This journey of PhD was possible with the consistent support and encouragement from my supervisors Dr. Paul M. Clarke and late Prof. Rory O' Connor. They both were pre-emptive in providing feedback, positive criticism and directing this research towards a fruitful outcome. I am grateful to them for being there and making this research a lifetime experience of learning and extending my horizon.

I am extremely grateful to the Dublin City University (DCU) and Lero - The Irish Software Research Centre, for their support, financial and otherwise, which made my PhD studies possible.

A constant source of motivation and strength for me in this journey is my mother (Mehboob) and late father (Abdul Rashid) who always had faith in me and kept me going through the trying times. I am thankful to both of them for being my guiding light and showering their unconditional love and affection on me.

I extend my thanks to Dr. Andrew MacCarren, Dr. Alessandra Mileo, and Dr. Marija Bezbradica in the School of Computing, DCU, for their valuable feedback that helped me to refine this work.

I sincerely thank Dr. Murat Yilmaz, for assisting with the supervision of the latter stages of my work.

A part of my PhD journey also comprised of friends, who made me laugh and lend an ear in time of need. I drove them crazy with my level of anxieties but they always found a way to be my true friends. I truly thank them all with the depth of my heart and cherish their constant presence during all those times.

Last but not the least I would like to thank anyone who one way or another became part of my PhD journey including the security personnel who ensured a secure work environment during the night times and staff members who helped to keep the office space clean.

Contents

Abstract . . . viii

Related Peer Reviewed Publications . . . ix

List of Figures . . . ix

List of Tables . . . xiii

List of Terms and Abbreviations . . . xv

1 The Focus of Research **1**

1.1 Introduction . . . 1

1.2 Motivation . . . 4

1.3 Research Objectives . . . 5

1.4 Research Hypothesis and Research Questions . . . 6

1.5 Research Process . . . 8

1.6 Thesis Structure . . . 9

2 Literature Review **11**

2.1 Introduction . . . 11

2.2 Background . . . 14

2.2.1 Open Source Software (OSS) . . . 14

2.2.2 Organisational Structure in OSS Projects . . . 16

2.2.3 Data, Information, and Knowledge . . . 21

2.3 Literature Review and Snowballing . . . 22

2.3.1 Details on Snowballing Procedure . . . 23

2.3.2 Initial Baseline Set of papers . . . 24

2.3.3 Inclusion/ Exclusion Criteria for Baseline Set . . . 25

2.4 Quality Assessment . . . 26

2.5 Iterations in Snowballing . . . 28

2.5.1 Backward Snowballing . . . 28

2.5.2 Forward Snowballing . . . 28

2.6 Application of Snowballing for Literature Review 29
2.6.1 Identifying the Baseline Set 29
2.6.2 Final Baseline Set 29
2.7 Iterations 31
2.7.1 Iteration 1 - Backward and Forward Snowballing 31
2.7.2 Iteration 2 - Backward and Forward Snowballing 32
2.7.3 Iteration 3 - Backward and Forward Snowballing 32
2.8 Data Synthesis 33
2.9 Discussion 35
2.9.1 Examining Knowledge Loss in OSS 35
2.9.2 Impact of Knowledge Loss in OSS projects 37
2.10 Reducing Knowledge Loss in OSS Projects 42
2.11 Manifestation of Knowledge in OSS Projects 47
2.11.1 Knowledge Creation 48
2.11.2 Knowledge Sharing 50
2.12 Knowledge Retention in OSS Projects 53
2.13 Chapter Summary 59

3 Research Methodology **61**
3.1 Introduction 61
3.2 OSS Project Structure 62
3.3 Philosophical Background 63
3.3.1 Positivism 64
3.3.2 Interpretivism 64
3.3.3 Pragmatism 65
3.3.4 Research Philosophy Adopted 66
3.4 Research Methodology and Methods 67
3.4.1 Quantitative Research 67
3.4.2 Qualitative Research 69
3.4.3 Mixed Methods Research 73
3.4.4 Research Design Adopted - Mixed Methods Research . . . 75
3.4.5 Data Analysis 77

3.4.6 Validity Concerns in Mixed Method Research 78
3.5 Empirical Research 79
3.6 Chapter Summary 82

4 Proactive Knowledge Retention Canonical Model 83
4.1 Introduction 83
4.2 Canonical Model Development Process 84
4.3 Selection of Data Components 85
4.3.1 Knowledge Retention in Organisations 86
4.3.2 Knowledge Retention Practices in Organisations 89
4.3.3 Knowledge Retention Mitigation Techniques and OSS Guides Online Resources 93
4.4 Data Preparation 94
4.5 Analysis - Principles of Grounded Theory 95
4.6 Application of Grounded Theory 98
4.7 Practices by Researcher 109
4.8 Canonical Model of Proactive Knowledge Retention in OSS projects 109
4.8.1 Communication 110
4.8.2 Contributor Motivation 113
4.8.3 Core Development Practice 115
4.8.4 Environment/ Ecosystem/ Culture 120
4.8.5 Governance and Leadership 123
4.9 Chapter Summary 128

5 Survey and Data Collection 130
5.1 Survey Development Process 130
5.2 Setting the Objectives 131
5.3 Survey design 132
5.3.1 Defining Target Population and Survey Sample 132
5.3.2 Conceptual Model of Survey 133
5.3.3 Data Collection Approach 134
5.3.4 Survey Instrument Design 134
5.3.5 Approaches for Data Analysis 135

5.3.6 Validity Considerations 136
5.4 Survey Instrument Development 137
5.5 Evaluating the Survey Instrument 139
5.6 Obtaining Valid Data 140
5.7 Analysing the Survey Data and Reporting 141
5.8 Conducting the Survey 142
5.8.1 Contributor Selection from GitHub 142
5.8.2 Selection of Projects 145
5.8.3 Plain Language Statement (PLS) 146
5.8.4 Sending Surveys using GMass 147
5.8.5 Phase - I: Pilot Survey 147
5.8.6 Phase - II: Survey 148
5.9 Chapter Summary 149

6 Data Analysis **151**
6.1 Data Analysis Overview 151
6.1.1 Overview of Survey Participants 151
6.1.2 Likert-Type Scale 154
6.1.3 Data Summary 155
6.2 Ranking Technique 160
6.2.1 Designing Practice Ranking Scheme 160
6.3 Categorical Ranking of Practices 164
6.3.1 Ranking of Practices Based on the Number of OSS Projects 165
6.3.2 Ranking Practices Based on Number of Years in OSS . . . 169
6.3.3 Ranking based on the Number of Years in Programming . 174
6.4 Ranking based on role type in OSS 180
6.4.1 Bug Reporter 184
6.4.2 Code Contributor 185
6.4.3 Maintainer 186
6.4.4 Reviewer 187
6.4.5 Committer 188
6.4.6 Document Writer and Editor 189

6.4.7 Tester . . . 191
6.4.8 Integrator . . . 192
6.4.9 Others . . . 193
6.5 Qualitative Data on PKR Model Completeness . . . 195
6.6 Chapter Summary . . . 196

7 Evaluation of Practices 197
7.1 Evaluating Practice Preference – Number of OSS projects . . . 198
7.2 Evaluating Practice Preference - Number of Years in OSS . . . 203
7.3 Evaluating Practice Preference - Number of Years in Computer Programming . . . 205
7.4 Evaluating Practice Preference - Different Roles in OSS projects . 208
7.5 Chapter Summary . . . 210

8 Conclusion 212
8.1 Research Overview . . . 212
8.2 Primary Impacts . . . 214
8.3 Recommendations for OSS projects . . . 216
8.4 Research Limitations . . . 218
8.5 Future Work . . . 220

Bibliography 222

Appendices 248

A Literature Review 248
A.1 List of Primary Studies . . . 248

B Proactive Knowledge Retention Canonical Model Development 254
B.1 Master Table Containing Data Components Linked to data Sources 254
B.2 Merging Conceptual Duplicates . . . 266
B.3 Primary Classification of Data Components . . . 286
B.4 Categorisation of Practices . . . 295
B.5 PKR Practices and Categories - Revisiting Change and Renaming 310
B.6 PKR Practices - First Review . . . 315

B.7 PKR Practices - Second Review 320
B.8 PKR Practices - Third Review 328
B.9 PKR Practices - Fourth Review 343
B.10 PKR Practices - Fifth Review 356

C The Survey Instrument and Data Collection **367**
C.1 Survey Instrument 367
C.2 Data Collected Through Survey Instrument 380
C.3 Data Collected Through Survey Instrument (Questions 5-13) . . 384
C.4 Data Collected Through Survey Instrument Questions 13-22 . . . 388
C.5 Data Collected Through Survey Instrument (Question 23-32) . . 392

D Data Analysis **397**
D.1 Description of Knowledge Retention Practices 397

Reducing Knowledge Loss in Open Source Software Projects

Mehvish Rashid

Abstract

From conception, to implementation, to deployment and to retirement, software development is a knowledge-intensive activity. To address the demands of knowledge acquisition and exchange, various mechanisms, processes and ceremonies have been established. These general knowledge acquisition and exchange challenges are amplified in Open Source Software (OSS) development where software developers are often volunteers, where their commitment to an OSS project may be transient, outside the reach of regular employment contractual arrangements, and in an environment where contributors may be widely geographically distributed. It is therefore the case that OSS projects present with very specific and highly demanding knowledge exchange challenges, and failure to address and manage these challenges can result in the lack of adoption, or worse the total collapse, of an OSS project.

This work is focused on identifying a set of practices that can be adopted in OSS projects to reduce knowledge exchange challenges, and this is important work for which no substantial earlier contribution exists. To develop a set of OSS Knowledge Retention (KR) practices, a variety of sources are incorporated, including knowledge management in general, software engineering based knowledge creation and adoption, and community feedback. Following a survey of OSS project contributors, a suite of 31 OSS KR practices is identified and ranked, though differences can be seen in terms of practice appreciation depending on OSS project experience level and role. This set of OSS KR practices can be adopted by OSS projects so as to reduce the knowledge exchange challenges in their projects, and to promote longer term OSS project viability and success.

Keywords: *Open Source Software*, *Knowledge Loss*, *Knowledge Management*, *Knowledge Retention*, *Software Engineering*.

Related Peer Reviewed Publications

1. Rashid, M., Clarke, P. M., & O'Connor, R. V. (2019). A mechanism to explore proactive knowledge retention in open source software communities. Journal of Software: Evolution and Process.

2. Rashid, M., Clarke, P. M., & O'Connor, R. V. (2019). A systematic examination of knowledge loss in open source software projects. International Journal of Information Management, 46, 104-123.

3. Rashid, M., Clarke, P. M., & O'Connor, R. V. (2018). An Approach to Investigating Proactive Knowledge Retention in OSS Communities. In European Conference on Software Process Improvement, pages 108-119. Springer, Cham.

4. Rashid, M., Clarke, P. M., & O'Connor, R. V. (2017). Exploring knowledge loss in open source software (OSS) projects. In International conference on software process improvement and capability determination, pages 481-495. Springer, Cham.

5. Rashid, M., Clarke, P. M., & O'Connor, R. V. (2017). Reducing Knowledge Loss in Open Source Software. In: Proceedings of the 13th International Symposium on Open Collaboration, pages 23-25, Galway, Ireland.

6. Rashid, M. (2016). Remedying knowledge loss in free/libre open source software. In Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, Article 4, pages 1-4. ACM.

List of Figures

1.1 Model of research process (adapted from (Oates, 2006)) 9
2.1 Mind map of the organisational structure in OSS projects. 21
2.2 Data - information - knowledge (Rhem, 2005) 22
2.3 Overview of literature review using snowballing 23
2.4 The detailed view of snowballing procedure to search papers . . . 24
2.5 Overview of literature review using snowballing 31
2.6 Applying snowballing process to search relevant papers on Knowledge Loss in OSS 33
2.7 The yearly distribution of papers 34
2.8 The distribution of papers according to the type of research conducted 35
2.9 Impact of knowledge loss in OSS projects 42
2.10 Knowledge Sharing Model (KSM) (Sowe et al., 2008) 51
2.11 Mind map of the knowledge creation and sharing in OSS projects 53
2.12 Mind map of mitigation approaches to knowledge loss in OSS projects 57
3.1 Onion model representing contributors in OSS projects users (Crowston et al., 2004; Crowston and Howison, 2005; Mockus et al., 2002) 63
3.2 Monomethod (1 and 8) and mixed-model designs (2 to 7) (Johnson et al., 2007) 74
3.3 The overall research methodology and road map for chapters 4-7 . 81
4.1 Development process of PKR canonical model in OSS projects . . 85
4.2 Pyramid of knowledge types and associated Knowledge Management (KM) strategies (Jansen, 2008) 88
4.3 8-step review process and outcome for every review with the number of categories and practices in PKR canonical model . . . 108

4.4 Proactive Knowledge Retention (PKR) model developed for OSS projects . . . 128
5.1 The process of survey development . . . 131
5.2 Process to extract survey participants from GitHub . . . 142
5.3 GitHub web page populated with projects with most stars . . . 143
5.4 Data relevant to contributors profiles on GitHub . . . 146
5.5 Data collected from the survey . . . 149
6.1 11-point Likert Scale used for conducting OSS community survey 155
6.2 The distribution of responses for practices (1-9) and frequency on the Likert scale (0-10) . . . 157
6.3 The distribution of responses for practices (10-18) and frequency on Likert scale (0-10) . . . 158
6.4 The distribution of responses on practices (19-31) and frequency on Likert scale (0-10) . . . 159
6.5 Distribution of responses obtained for 31 practices based on Median, Mean, Std (SD), AvgDev (MAD) . . . 161
6.6 Ranking pyramid - the overall ranking of practices . . . 164
6.7 The disparity among three categories of contributors based on the number of OSS projects and values obtained for mean, median, SD, and MAD . . . 165
6.8 Hierarchy of practices based on the number of OSS projects . . . 169
6.9 The disparity among three categories of contributors based on the number of years OSS and values obtained for mean, median, SD, and MAD . . . 170
6.10 Hierarchy of practices based on the number of years in OSS . . . 174
6.11 The reported effectiveness of practices based on number of years in computer programming . . . 175
6.12 Hierarchy of practices based on the number of years in computer programming . . . 180
6.13 Mean values by role type . . . 181
6.14 Median values by role type . . . 182
6.15 SD values by role type . . . 183

6.16 MAD values by role type 184
7.1 The practice preference for contributors with experience based on the number of OSS projects 198
7.2 Ranking of practices based on contributors with an experience of 1-5, 5-10, 10+ in OSS projects 201
7.3 Preference trend of contributors on the agreement among three categories of contributors 204
7.4 Ranking of practices on their effectiveness by contributors with varying experience in programming 206
7.5 The ranking trend of practices by contributors performing tasks in varying roles in OSS projects. 208

List of Tables

2.1 Five search strings to extract initial baseline set in SB 25
2.2 Quality assessment criteria to evaluate papers 27
2.3 Representation of knowledge retention techniques with knowledge conversions type and codification strategy 44
2.4 Summary of Research focus on KM relevant activities in OSS (Crowston et al., 2012) 46
3.1 Three philosophical concerns 66
3.2 Summarising three types of research methodologies 78
4.1 Master table with data components and link to sources 94
4.2 Example of merging conceptual duplication 99
4.3 Example of primary classification 100
4.4 Example of categorising classification of practices 101
4.5 PKR practices and relevant category 102
4.6 PKR practices the first review 103
4.7 PKR Practices the second review 104
4.8 PKR Practices the third review 105
4.9 PKR Practices the fourth review 106
4.10 PKR Practices the fifth review 107
5.1 Conceptual model of survey 134
6.1 Representation of OSS contributors profile in the survey population 152
6.2 Overview of the survey data obtained 156
6.3 Ranking of overall practices 163
6.4 Categorical ranking of practices based on 1-5 numbers of OSS projects 166
6.5 Categorical ranking of practices based on 5-10 numbers of OSS projects 167

6.6 Categorical ranking of practices based on 10+ numbers of OSS projects . . . 168
6.7 Categorical ranking of practices based on 1-5 numbers of years in OSS . . . 171
6.8 Categorical ranking of practices based on 5-10 numbers of years in OSS . . . 172
6.9 Categorical ranking of practices based on 10+ numbers of years in OSS . . . 173
6.10 Categorical ranking of practices based on 1-2 numbers of years in programming . . . 176
6.11 Categorical ranking of practices based on 2-5 numbers of years in programming . . . 177
6.12 Categorical ranking of practices based on 5-10 numbers of years in programming . . . 178
6.13 Categorical ranking of practices based on 10+ numbers of years in programming . . . 179
6.14 Ranking of practices based on bug reporters in OSS projects . . . 185
6.15 Ranking of practices based on code contributors in OSS projects . 186
6.16 Ranking of practices based on maintainers in OSS projects 187
6.17 Ranking of practices based on reviewers in OSS projects 188
6.18 Ranking of practices based on committers in OSS projects 189
6.19 Ranking of practices based on document writers and editors in OSS Projects . . . 190
6.20 Ranking of practices based on testers in OSS projects 191
6.21 Ranking of practices based on integrators in OSS projects 192
6.22 Ranking of practices based on others in OSS projects 193
6.23 Ranking of practices based on nine roles in OSS projects 194
D.1 Practice description against Practice No. 398

List of Terms and Abbreviations

BSB Backward Snowballing. 28, 29, 31, 32, 218

CSS Closed Source Software. 15–17, 19, 35, 36, 40, 41, 46, 49, 57–59

FSB Forward Snowballing. 22, 24, 28, 29, 32, 218

KL Knowledge Loss. 3–6, 9, 11–14, 22, 26, 30, 35–37, 41, 42, 45, 47, 53, 54, 57–60, 66, 75, 79, 86, 93, 95, 131, 218

KM Knowledge Management. xiii, 42, 43, 45, 46, 59, 60, 89, 91, 114

KR Knowledge Retention. 42–46, 57, 58, 60, 66, 80, 82, 86, 89, 93–95, 113, 116, 138, 199, 214, 216, 217, 220, 221

OSS Open Source Software. 1–4, 9–12, 14, 15, 24, 26, 40, 51, 132, 138, 142

PKR Proactive Knowledge Retention. x, xi, 9, 77–79, 82, 84, 85, 93–96, 101, 102, 104, 107–110, 128, 131–134, 137, 138, 141, 150, 213

QDA Qualitative Data Analysis. 96

SB Snowballing. 22, 23, 25, 27–29, 32, 33, 48, 59, 218

Chapter 1

The Focus of Research

1.1 Introduction

Open Source Software (OSS) is enlisted as one of the four core components of Open Science (OS), which relates to the movement that provides free accessibility to scientific research data and its dissemination to all levels of inquiring society (Pontika et al., 2015). OSS is a broad term used to embrace software developed and released under an "open source" license allowing technically inclined users free access to the code, to inspect, modify and redistribute the software (Crowston et al., 2004, 2006). In 2001, European Commission (EC) used the term FLOSS, Free/Libre Open Source Software, for the first time to avoid taking sides on debate on the distinction between free software and open-source software. OSS is a term used to identify software developed and released under an "open source" license that complies with Open Source Definition (OSD). The OSD uses either a short definition based on four criteria as in the Free Software Foundation (FSF) or a more extended version based on ten criteria as in the Open Source Initiative (OSI). The difference between these two definitions is only of language while underlying meaning and outcome is the same. "The freedom to use, change, sell, or give away the software, the availability of source code, and the protection of authors' intellectual property rights are the central tenets of the OSD" (Feller et al., 2002). Users with the technical inclination can use, freely access the code, inspect, modify and redistribute the software (Crowston et al., 2006). However, the freedom to use source code from an OSS project and its distribution vary

based on which category of OSS license agreement is applied.

There are three main categories of OSS licenses based on their degree of restrictiveness: Strong-copyleft, weak-copyleft and non-copyleft (Subramaniam et al., 2009). A strong-copyleft or restrictive license requires that any derivatives of the original software are also licensed in a similar manner. Weak-copyleft licenses allow the derivatives of the software to be released under a different license. Non-copyleft licenses allow the software, including derivatives, to be redistributed under a different license than the original one. While free software mostly identifies with GNU Public License (GPL), OSS license agreements may vary based on the incorporation of the software that can be either propriety or free.

Another term to represent free software is Free Open Source Software (FOSS). The term "free" in FOSS was not considered by some to adequately express the notion of freedom and consequently, in 2001, the European Commission (EC) introduced the term Free/Libre Open Source Software (FLOSS), to avoid taking sides in the debate and to stay neutral on the distinction between free software and open-source software.

This research refers to the term OSS with a focus on projects that comply with OSD, with the purpose of not excluding projects based on their license restrictions and the types of existing collaborations, commercial involvement, or volunteer based. The objective is to investigate projects operating under the umbrella of OSS irrespective of the organisational setup, design a mechanism to reduce Knowledge Loss (KL).

The importance of OSS is of vital importance in our daily work routine and there are thousands of OSS projects operating worldwide, including Linux operating system, Apache Web Server, Mozilla Firefox, OpenOffice, Perl, Python, GCC, Android and Chrome by Google, and many more (Rashid et al., 2019b). There has been an exponential rise in OSS products and their use, as indicated by 430,000 projects hosted in 2014 on the SourceForge portal (Silic

and Back, 2017). The projects are of varying sizes and at times involve commercial firms that heavily depend on OSS system (Crowston et al., 2012). During 2014 Google's 2014 Google's open source mobile Android operating system had about one billion users across all devices (Van der Meulen and Rivera, 2014). A survey conducted in 2015 reported that almost 78% of companies run operations on OSS and 66% of companies have incorporated open source software to create products for customers (Software, 2015).

The term OSS in this research is applicable to a vast variety of projects complying with OSD. This research investigates the problem of KL in OSS projects due to contributor turnover. The phenomenon where contributors working in project teams join, leave, or change their role is referred to as 'turnover' (Foucault et al., 2015). The turnover can be catastrophic for the project if a contributor who is knowledgeable on major parts of the system leaves and this reduces the spread of the knowledge (Donadelli, 2015). In OSS projects, a smaller percentage of core contributors about 20% perform 80% of coding tasks. Contributor turnover is rated as being very high both in the software industry (Zhou, 2009) and in OSS projects (Foucault et al., 2015; Otte et al., 2008; Rigby et al., 2016; Robles and Gonzalez-Barahona, 2006) and mitigating its effects is considered a significant problem (Fronza et al., 2013). Turnover is inevitable due to the transient nature of OSS project workforces (Michlmayr, 2007a; Xu, 2006; Yu et al., 2012) causing KL (Izquierdo-Cortazar et al., 2009; Rigby et al., 2016). KL due to contributor turnover, is the loss of experience and expertise in projects. Contributor turnover impacts software quality (Foucault et al., 2015; Mockus, 2010) and productivity (Izquierdo-Cortazar et al., 2009; Schilling et al., 2011), which might also threaten the overall sustainability of projects.

1.2 Motivation

The teams in projects evolve due to contributors who are constantly joining, leaving, or changing their role in the project. In many large projects, a high turnover has been observed leading to the formation of the new development teams (Robles and Gonzalez-Barahona, 2006). KL impacts the productivity of the OSS projects in two ways:

1 *The effort required to acquire knowledge to perform the maintenance tasks*

2 *The loss of effort when code is orphaned and removed from the project*

In order to write quality software, code-knowledgeable contributors are required. Searching knowledge is argued to be time consuming and costly (Von Krogh et al., 2005). The search efforts can vary depending on the source and the level of details. A post or a query on the project mailing list requires less effort while searching through the results of a search engine or examining the clues into source code documentation is perhaps more time consuming (Von Krogh et al., 2005). A study on the GNOME project reported that 30 months' time is needed for the contributor to understand the software code and to make a contribution (Herraiz et al., 2006). Developers gradually become productive taking more than a year's time on a project to reach a productivity plateau (Zhou and Mockus, 2010). The time to complete distributed tasks is estimated to be three times longer than for co-located tasks (Zhou and Mockus, 2010). The time required by a new person to learn the inner workings of the project when experienced contributors leave, causes considerable productivity loss (Izquierdo-Cortazar et al., 2009). In-depth understanding of software code and interconnecting file structure is not required to complete simple tasks. On the other hand, contributors may have difficulty performing non-trivial tasks due to 'information blocking', i.e. unavailability of the relevant information to complete a task (Izquierdo-Cortazar et al., 2009). The productivity of the contributor and overall project suffers due to the information

blocking and a lack of understanding of the code base. According to estimates, information blocking consumes 60% of developers efforts (Liu et al., 2005).

During the preparation of a release, contributors make changes to align their work with the goals of the release (Michlmayr, 2007a). As abandoned code increases on the project, the numbers of reported defects increase as well (Otte et al., 2008). The maintenance of abandoned code is difficult because the team lacks knowledge of its creation and structure (Herbsleb and Mockus, 2003). The source code that remains unmaintained (unless a stable legacy system) has an element of uncertainty for the development team since the contributors who wrote it have left the project (Izquierdo-Cortazar et al., 2009). Removal of unmaintained code results in loss of existing functionality and may impact users of the system (Michlmayr, 2007a).

1.3 Research Objectives

The objective of this research is to systematically investigate KL and mitigate its implications in OSS projects. The goal is to reduce KL by identifying knowledge retention best practices through systematic study and by engagement with practitioners. "Knowledge is information combined with experience, context, interpretation and reflection" (Davenport et al., 1998). Knowledge generation is continuous through the process of knowledge creation and sharing in OSS projects and is cyclic in nature. There are two kinds of knowledge generated, namely tacit and explicit, tacit knowledge being that has not been made explicit, that may for example be in the mind of one or more individuals but not documented (Ryan and O'Connor, 2013). Both tacit and explicit knowledge are inevitably created as software is produced, and the advent of agile software development (Beck et al., 2001) has placed an emphasis on reducing explicit documented knowledge.

The impact of this development suggests a heightened demand to address

tacit knowledge retention in software development projects and especially on OSS projects where contributor continuity is unpredictable and where contributors might not have face-to-face communication opportunities on a regular basis but only through asynchronous means of communication facilitated by technology-mediated channels. The KL due to the loss of experience and expertise on the project impacts productivity and additional time is required to learn the workings of the project when original contributors are no longer accessible (Izquierdo-Cortazar et al., 2009).

1.4 Research Hypothesis and Research Questions

The central hypothesis of this work states that:

> Knowledge retention practices can be adopted towards effectively addressing knowledge loss in OSS projects.

The reactive approach of knowledge retention refers to enablement of knowledge transfer activities when the contractually bonded employee is leaving the organisation. In OSS projects, inevitable turnover due to transient nature of contributors (Michlmayr, 2007b; Yu et al., 2012) (Yu et al. 2012; Michlmayr 2007; Xu 2006) and absence of contractual bindings for notification before contributors leave, make it difficult to enable any reactive knowledge transfer activity. Therefore, the reactive approach of knowledge retention that may be practised in a traditional software organisation may prove wholly ineffective for OSS projects. The central hypothesis stresses the need for proactive knowledge retention practices in OSS projects arising from the difference of organisational and governance structure between OSS and traditional software organisations, e.g. absence of contractual agreement between employer and employee. The proactive approach in OSS projects is to retain knowledge while contributors are part of the project. The central hypothesis leads to the first research question:

RQ1. What is the existing state-of-the-art literature on knowledge loss due to turnover in OSS projects?

OSS projects might be considered an extreme example of globally distributed projects with transient contributors performing tasks in different roles. The characteristics pertaining to the OSS development come in different flavours and vary from project to project. OSS projects either can have commercial involvement with stakeholders who strategically employ contributors to participate in development of software to be later on attributed to a product or can be purely community based with contributors mostly volunteers participating towards software development. In the former, the work settings resemble to the ones in the development of propriety software, while in later the work settings are similar to horizontal hierarchy with informal management and self-assigned tasks. The challenge is to formulate the proactive knowledge retention, which can resonate with the idiosyncratic nature of OSS projects without causing an overhead to the productivity of the project and contributors. The OSS communities focus on self-direction and favour intrinsic value system and therefore imposing a strategy that calls for the contributor to give away their freedom to choose freely will cause deviation from the enthusiasm of doing well for the society. The next step ahead is to identify knowledge retention practices and evaluate on their effectiveness in OSS projects work settings, which leads to the following research question:

RQ2. What are the effective knowledge retention practices in OSS projects?

In order to investigate RQ2, the following two sub-questions need to be answered, which are iterated as follows:

RQ2.1 How can a comprehensive set of knowledge retention practices be developed for OSS projects?

RQ2.2 How can OSS knowledge retention be evaluated in OSS projects?

1.5 Research Process

The research process for this work is adapted from the process model presented by (Oates, 2006). In the model, research questions formulate based on the researcher‘s motivation and by conducting the literature review. In order to investigate research questions a research strategy is identified. In the next step, a data generation method is aligned with the research strategy, traceable to research questions. Selecting right method for data generation ensures that collected data is suitable to investigate the research questions. Final step presented by Oates (2006) model of research process is data analysis. Figure 1.1 entails the four step research process representing the work in entirety. The outcome of data analysis marks the end of research process by reporting research findings and conclusions. The description for each step follows:

1 *Research initiation - involves identification of concepts and research questions based on the motivation and related literature review.*

2 *Research strategy - Identifying an approach to conduct research on the identified questions.*

3 *Data generation - Formulating a reliable approach for data collection to satisfy research requirements and to obtain valid results for research questions*

4 *Data analysis and evaluation - Identifying suitable methods for data analysis and presenting evaluations.*

The research findings and conclusions are the final outcome of the data analysis.

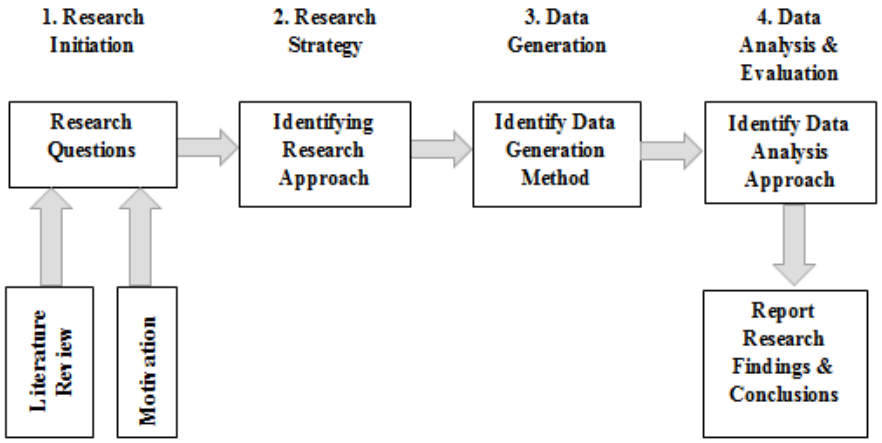

Figure 1.1: Model of research process (adapted from (Oates, 2006))

1.6 Thesis Structure

This thesis comprises of the following chapters.

Chapter 2 presents a systematic examination of the state-of-the-art literature on the phenomenon of KL in OSS due to contributor turnover using snowballing as a search strategy. The literature review elaborates on problem identification and details the impact of KL in OSS projects and discusses mitigation techniques to reduce KL in OSS projects.

Chapter 3 discusses the research methodology designed to investigate and to devise a set of PKR practices for OSS projects. The chapter describes philosophical concerns, emphasises research methodologies including research design, research methods, and selection of a suitable methodology for empirical investigations in this work.

Chapter 4 elaborates on the development process of Proactive Knowledge Retention (PKR) practices for OSS development environment. The PKR practices are derived from literature inOSS and traditional software organisations, which are referred as data sources. In order to develop the PKR canonical model, the candidate details are selected from data sources referred as data components, which go through a rigorous process to consolidate PKR

practices by applying the principles of grounded theory.

Chapter 5 entails the process of designing the survey instrument and conducting the survey for data collection. The goal of conducting the survey is to evaluate the effectiveness of PKR practices by engaging with OSS practitioners. The OSS contributor profiling in the survey sources rich data from contributor evaluations on the effectiveness of proactive knowledge retention practices in OSS projects.

Chapter 6 details the techniques for data analysis that result in the ranking of practices. Extensive data analysis incorporates the profiling of contributors based on their experience in OSS projects. The preference trend of contributors is described in detail with the identification of practices that are ranked as most effective.

Chapter 7 entails the difference of opinion on the effectiveness of particular practices among various sub-groups of OSS contributors. The evaluation of practices in this chapter revolves around the disparity of practice ranking among the experienced, middle level experienced and inexperienced group of contributors in OSS projects. Contributors classify based on their contributions to the number of OSS projects, contributions by the number of years in OSS projects, on the number of years in computer programming, and by different roles in OSS projects.

Chapter 8 summarises the research conducted to answer research questions, research methodology adopted, and the main findings. The primary impacts of the research and recommendations for OSS projects are also discussed. The limitations in this research along with and future work are also articulated.

Chapter 2

Literature Review

2.1 Introduction

This chapter discusses the literature review conducted to explore the phenomenon of Knowledge Loss (KL) in OSS projects. The methodology followed to conduct the literature review and its results have already been accepted through peer-review (Rashid et al., 2019b). The text that follows in this chapter essentially mirrors the content of this published material, and includes the knowledge and substantive findings on the subject.

OSS development is known to be a knowledge focused activity which relies heavily on contributors who can be volunteers or paid workers and are geographically dispersed. While working on OSS projects contributors acquire project related individualistic knowledge and gain experience and skills, which can remains unshared with others and is usually lost once contributors leave a project. Most of the software development organisations face the problem of KL as employees leave, but this situation is exasperated in OSS projects where contributors can be volunteers with largely unpredictable engagement duration. Contributor turnover is inevitable due to the transient nature of OSS project workforce causing KL, which threatens the overall sustainability of OSS projects and impacts negatively on software quality and contributor productivity. The objective of literature review is to deeply and systematically investigate the phenomenon of knowledge loss due to contributor turnover in OSS projects as presented in the state-of-the-art literature and to synthesise the information

presented on the topic. Furthermore, based on the learning arising from the investigation it is the intention to identify mechanisms to reduce the overall effects of KL in OSS projects.

Software development is a knowledge-intensive activity, which involves intense complexity (Clarke et al., 2016). OSS has had a profound impact on the way in which software is developed and consequently on the perception of software development (Hagan et al., 2007). OSS is one of the representative examples of open collaboration (Lee et al., 2017). In OSS projects, contributors can be volunteers or paid workers who participate in software development activities. While working on OSS projects contributors acquire project related knowledge and gain experience and skills. Examples of knowledge that are required to accomplish software development tasks on projects include application domain, system's architecture, use of particular algorithms to code, insights into requirements, programming language and development environment (Anquetil et al., 2007). Valuable individualistic knowledge, which remains unshared with others, is lost once contributors leave the project. Organisations constantly face the problem of KL as employees leave (De Long and Davenport, 2003; Jennex and Durcikova, 2013; Viana et al., 2015), a situation which is perhaps exasperated in OSS projects (Izquierdo-Cortazar et al., 2009; Rigby et al., 2016) where most contributors are volunteers with largely unpredictable engagement durations (Robles et al., 2005). The phenomenon of volunteers joining and leaving at their discretion is more common in OSS projects than with hired employees in Closed Source Software (CSS) (Robles et al., 2005). Contributor attrition leads to KL in OSS projects.

The phenomenon where contributors working in project teams join, leave, or change their role is referred to as 'turnover' (Foucault et al., 2015). The turnover can be catastrophic for the project if a contributor who is knowledgeable on major parts of the system leaves and this reduces the spread of the knowledge (Rigby et al., 2016). Contributor turnover is rated as being very high both in the software

industry (Zhou, 2009) and in OSS projects (Foucault et al., 2015; Otte et al., 2008; Rigby et al., 2016; Robles and Gonzalez-Barahona, 2006) and mitigating its effects is considered a significant problem (Fronza et al., 2013). Turnover is inevitable due to the transient nature of OSS project workforces (Michlmayr, 2007a; Xu, 2006; Yu et al., 2012) causing KL (Izquierdo-Cortazar et al., 2009; Rigby et al., 2016). KL is the loss of experience and expertise in OSS projects, which not only impacts software quality (Foucault et al., 2015; Mockus, 2010) and contributor productivity (Izquierdo-Cortazar et al., 2009; Schilling et al., 2011), but which might also threaten the overall sustainability of OSS projects.

OSS projects are constantly evolving as indicated in the adaptated staged model for OSS systems (Capiluppi et al., 2007), and maintenance plays a significant role in project evolution (Lin et al., 2017; Rigby et al., 2016). As asserted “it is harder to separate out maintenance and development since they tend to occur together" (Michlmayr, 2007a). A system that stops evolving is an indication that it may become a legacy in the near future (Capiluppi et al., 2007). KL in this work refers to the loss of experience and expertise in OSS projects that can result in the decline of evolution on OSS systems (Joblin et al., 2017).

To gain an insight into the phenomenon of KL in OSS projects due to turnover, a literature review is conducted. The research question is structured using the PICOC (Population, Intervention, Comparison, Outcome and Context) criteria to structure research questions for a Systematic Literature Review (SLR) (Petticrew and Roberts, 2008), with only population, intervention, and outcome being relevant for the research question:

- *Population: Open Source Software and associated synonyms.*
- *Intervention: Knowledge loss and turnover.*
- *Outcome: Existing themes and patterns related to knowledge loss and turnover in OSS projects*

The related research question is:

> What is the existing state-of-the-art literature on knowledge loss due to turnover in OSS projects?

The remainder of the chapter is structured as follows: Section 2.2 provides an insight into the set up of OSS projects, work structure and knowledge relevant concepts. Section 2.3 presents the literature review methodology, section 2.4 details quality assessment, section 2.5 discusses the concept of iterations in snowballing and section 2.6 presents the application of the literature review methodology. The details on iterations and selection of paper is given in section 2.7. Section 2.8 elaborates on data synthesis. Section 2.9 articulates a discussion on problem identification and details the impact of KL in OSS projects. Section 2.10 discusses concept of knowledge retention. Section 2.11 discusses the manifestation of knowledge in OSS projects. Section 2.12 comprises of mitigation techniques to reduce KL in OSS projects.

2.2 Background

2.2.1 Open Source Software (OSS)

OSS is enlisted as one of the four core components of Open Science (OS), which relates to the movement that provides free accessibility to scientific research data and its dissemination to all levels of inquiring society (Pontika et al., 2015). OSS is a term used to embrace software developed and released under an "open source" license that complies with Open Source Definition (OSD). The OSD uses either the shorter version based on a four point criteria as in Free Software Foundation (FSF) or the longer version based on a ten point criteria as in Open Source Initiative (OSI). The difference between the two definitions is one of language while the underlying meaning and outcome is the same. "The freedom to use, change, sell or give away the software, the availability of source code and

the protection of authors' intellectual property rights are the central tenets of the OSD" (Feller et al., 2002). Users with a technical inclination can use, freely access the code, inspect, modify and redistribute the software (Crowston et al., 2004, 2006). However, the freedom to use and distribute source code from OSS varies based on which category of OSS license agreement applies (Scacchi, 2007). There are two main categories of OSS licenses: copyleft and non-copyleft (DeBrie and Goeschel, 2016). Copyleft or a restrictive license requires that any work based on copyleft license, when incorporated with other work, stays under the copyleft license. Further, the complete code with modifications and the notice of modifications and attributions is accessible to the user free of cost. Examples of restrictive licenses are GNU General Public License (GPL versions 2 and 3), the Affero GPL (AGPL), and Mozilla Public License (MPL) and the Lesser GPL (LGPL). Less restrictive licenses such as LGPL and MPL, do not strictly adhere to OSD criteria and are considered as weak copyleft licenses. Non-copyleft or permissive licenses allow the distribution of software without any source code or modifications and provide a royalty free license to use the software for commercial purpose. Permissive licenses are useful for businesses who intend to sell the product as propriety software. Examples of permissive licenses are the Massachusetts Institute of Technology (MIT) License, Apache License 2.0, W3C, and the BSD License. While free software mostly identifies with GPL, OSS license agreements may vary based on the incorporation of the software that can be propriety or free. Another term used to represent free software is Free Open Source Software (FOSS). The term "free" in FOSS fails to express the notion of freedom and becomes invisible, with the main stress on OSS. In 2001, the European Commission (EC) used the term Free/Libre Open Source Software (FLOSS), for the first time to avoid taking sides in debate and stay neutral on the distinction between free software and open-source software. Three differences exist between OSS and CSS or commercial software (Capiluppi et al., 2007). The first concerns the availability of commercial

software releases to third parties only when it is complete, running, fully tested and authorised (Capiluppi et al., 2007). Conversely, OSS systems are available before the first official release while still in versioning system repositories, and at any instant might be downloaded (Michlmayr, 2007b). The second difference of OSS to CSS, relates to the presence of an evolution stage (on going software development) in OSS after the start of the servicing stage (no new functionalities are added but fixes are performed to existing system). The third difference noticed was the transition of the OSS from a phase out (no fixes of existing functionalities or addition of new ones) to evolution.

Software maintenance is an ongoing activity in software development, comprising several types of activities relating to fixing faults in the system, adapting the system to changes in the operating environment and adapting the system to changes in the original requirements (Basili, 1990). Software maintenance is the field which is concerned with the evolution of a software system after its initial release (Michlmayr, 2007b). In OSS, maintenance and development (or evolution) are not considered as two separate phases of the software development cycle (Michlmayr, 2007b). Maintenance in an OSS project is compared to reinvention (Crowston et al., 2012), which is "a continuous source of adaptation, learning and improvement in OSS functionality and quality" (Scacchi, 2003). Overall maintenance activities in OSS projects are ongoing along with the evolution of the system.

2.2.2 Organisational Structure in OSS Projects

The development in OSS projects is distinguished from CSS or traditional software by the usage of the terms: 'cathedral' and 'bazaar' (Raymond, 1999). In the cathedral, control is with one main person who controls progress. On the contrary, in the bazaar, the contributor decides when to contribute to a project (Crowston et al., 2004). In the cathedral phase, software development starts with a smaller group of developers and the source code is not shared or

accessible to users. While in the bazaar phase, software development has a large number of volunteers who can access the source code and contribute such as adding new requirements, bug fixes, and reporting defects. It is argued that a typical OSS project starts with a cathedral development style and then transitions to a bazaar development style and cathedral and bazaar phases are not considered mutually exclusive in OSS development (Capiluppi and Michlmayr, 2007).

In OSS, each project is considered similar to an organisation as in software industry or CSS. The layered structure called an onion model represents the organisational structure in the OSS community (Crowston and Howison, 2005; Dinh-Trong and Bieman, 2004). The teams in OSS have a hierarchical onion-like structure, consisting of core, co-developer, active users and passive users. The core is a small group of contributors who are responsible for most of the code and ensure the design and evolution of the project. Co-developers contribute by reviewing or modifying the code or providing bug fixes. Active users use the recent release and contribute bug reports or feature request but do not contribute code. The farthest group of members from core is passive users and their number is difficult to predict (Crowston et al., 2004). The representation of contributors in OSS projects can deviate from the above presented onion model. For instance, Linux developers organise themselves into two groups, 'core' and 'periphery' (Lee and Cole, 2003). The core consists of the project leader and hundreds of maintainers. Periphery is a large group of developers further divided into two teams: development and bug reporting. Core contributors go through all roles starting as a user and progressing to be among the core group of contributors (Ye et al., 2008). There can be different paths that may be followed to become a core contributor. It is reported that volunteers joining the OSS project follow the onion model and climb the ladder to become a core contributor based on the meritocracy, while hired developers are integrated into the project faster (Herraiz et al., 2006).

OSS project collaborations can be of three types: community-based, non-profit organisation and commercially based. Community projects are online projects where contributors participate voluntarily and contribute their time as is suitable for them (Xu, 2006). Volunteers collaborate on OSS projects during their free time without profiting directly by any economic incentives for their efforts (Robles et al., 2005). The motivation for volunteers to participate in OSS projects is to learn new skills, make code contributions and become known within the OSS community, which might pave the way to future career opportunities (Crowston, 2011). Volunteers drive on intrinsic motivation, which relates to the feeling of satisfaction, competence, and fulfilment from code writing (Schilling et al., 2011; Xu, 2006). There is also an intrinsic motivation for the knowledge provider such as altruism or learning by helping others solve problem. In OSS, it is hard to predict when a volunteer will leave a project. This unpredictable nature of commitment from volunteers creates an element of risk within OSS projects and managing volunteer contributors can cause certain problems not evident in traditional software development (Robles and Gonzalez-Barahona, 2006). The development of free software slows or stalls in the absence of the lead developer who initiated the project or when a contributor decides to stop working on the project. Community open source projects take their organisational form from an Internet-based community, and the developers are mostly the volunteers (Lee and Cole, 2003). For instance, volunteers who manage the Apache project are in fact developers with a full-time job who work part-time on the Apache project. Debian is a project in which all contributors are volunteers (Robles et al., 2005). The tasks performed by volunteers in Debian include maintaining software packages, supporting the server infrastructure, developing Debian-specific software, for instance, the installation routine and package management tool, translating documentation and Web pages.

In non-profit organisation projects, developers are either paid workers or

volunteers. The project is mature enough and is funded as a formal organisation. There is still some element of a community project maintained in such projects, for example, the Apache Software Foundation (Xu, 2006). In commercially involved projects, a software company sponsors projects and employs the majority of contributors. A commercial company, Netscape, managed the Mozilla project in the past. The developers hired for the project worked full-time and for pay (Dinh-Trong and Bieman, 2004). Companies like IBM, HP, SUN (now acquired by Oracle), sponsor OSS projects in which major contributors are paid developers (Fitzgerald, 2006). The organisational structure of OSS projects is considered to be highly dynamic as compared to CSS or traditional software development organisations (Jensen and Scacchi, 2005). Furthermore, the contributors in the OSS projects are transient in nature. While Internet-based knowledge communities are great avenues for contributors to learn and expand their skill sets, it is argued that they influence job hopping behaviour due to the availability of more opportunities, since it sends a strong signal to a potential employer of the contributors expertise and skills (Huang and Zhang, 2013). OSS projects can be a hub of innovation and evolving ecosystems with the involvement of commercial organisations (Capiluppi et al., 2012; Crowston et al., 2012) which align their strategic goals with the product development in OSS projects. OSS communities operate on globally distributed and virtual environments using the Open Source Software Development (OSSD) model (Jensen and Scacchi, 2005). The development in OSS projects is independent, self-assigned and in parallel streams without much coordination due to geographical dispersion (Michlmayr, 2007b). In OSS, tasks are not assigned and contributors make contributions based on their interest and discretion (Mockus et al., 2002).

The collaboration and communication is through asynchronous means, facilitated by technology-mediated channels (Sharif et al., 2015; Sowe et al., 2008; Vasilescu et al., 2014). In an informal, loosely defined community,

reciprocity is highly relevant to OSS development (Kuk, 2006). It is argued that interaction is of a strategic nature between individual developers and highly resourceful developers. This leads to the formation of smaller but better organized structures in OSS development (Kuk, 2006). Some measures suggested to gauge the impact of knowledge sharing communities include participation inequality, conversational interactivity, and cross-thread connectivity (Kuk, 2006). Participation inequality, if it exists to a certain level, has a positive impact on knowledge sharing. Moreover, strategic interaction expands knowledge sharing through reciprocal and overlapping activities (Kuk, 2006).

Epistemic interactions in OSS development involves a balancing act between exploration and exploitation activities (Lee and Cole, 2003). As an example, the successful Linux kernel development project has adopted a simple two-tier structure for interaction through mailing lists. The first tier consists of the peripheral mailing lists useful for innovation and for criticisms to encourage learning and quality control. The second tier consists of the core mailing lists, helpful for selecting and retaining the features, applications, and codes for the future incremental additions to the next product release (Kuk, 2006).

Altogether, OSS projects have an organizational outlook of an extreme example of large-scale globally distributed software engineering (GSD) (Joblin et al., 2017; Mockus et al., 2002). In such a dense environment where interaction and networking among community members is complex and spreads across various online communities, contributor departure can cause serious damage to the OSS project. Figure 2.1 summarises the organisational structure of an OSS project as a mind map. In the parenthesis reference to the respective study is given, which are listed under primary studies (e.g. PS1, PS2, PS3 and so on) in appendix A.1.

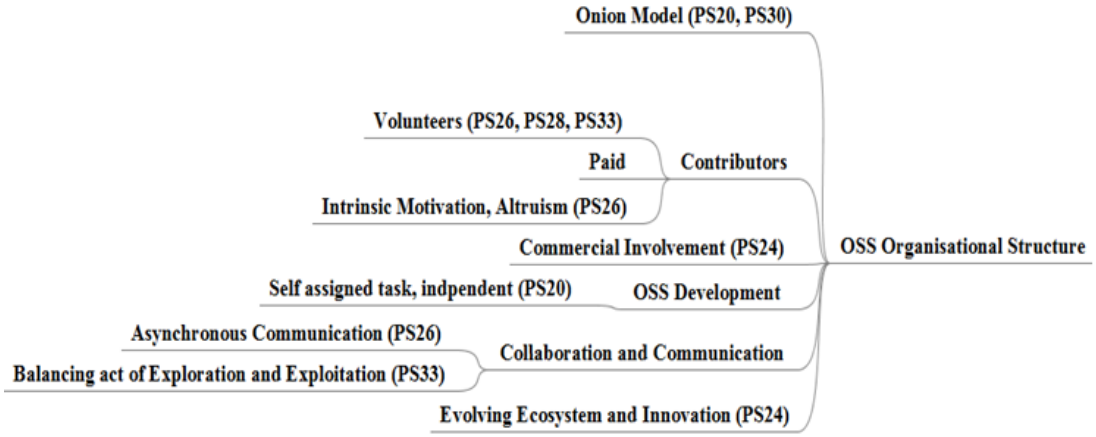

Figure 2.1: Mind map of the organisational structure in OSS projects.

2.2.3 Data, Information, and Knowledge

It is important to clarify the relationship among "data", "information", and "knowledge". Data can be considered the raw material for information, which, in turn, can be considered the raw material for knowledge (Zins, 2007). Data represent observations and facts without any context or meaning and information is the result of associating data in a meaningful context (Zack, 1999b). In order to convert data into information, it must be contextualised, categorised, calculated and condensed (Davenport et al., 1998). There are different views on the definition of knowledge. Nonaka in (Nonaka, 1994), defines knowledge as "a justified belief that that increases an entity's capacity for effective action". Schubert et al., using state of mind perspective, "define knowledge by emphasising, knowing and understanding through experience and study" (Schubert et al., 1998). This research refers to knowledge as experience and expertise built by a contributor that evolves from the day-to-day interaction on OSS project.

Knowledge is driven from information (Davenport et al., 1998) Knowledge is the product of an individual's experience and accumulates because of communication or inference (Zack, 1999b). Knowledge storage is argued to be impossible and it is asserted that information leads to knowledge transformation

(Aggestam et al., 2010). The stored information based on the individual experience is utilised to make decisions. The conversion of data, information, and knowledge is depicted in Figure 2.2.

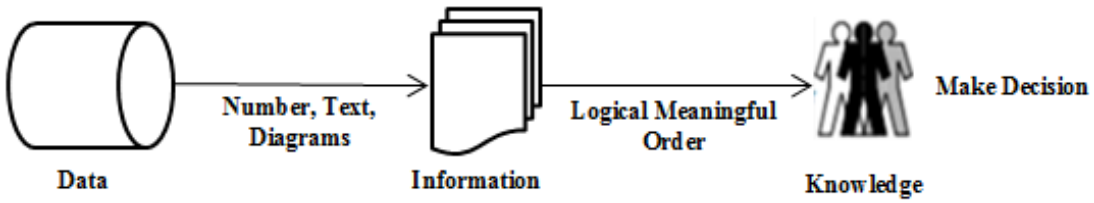

Figure 2.2: Data - information - knowledge (Rhem, 2005)

2.3 Literature Review and Snowballing

In order to find the relevant literature on the topic of KL in OSS, the literature review was discharged using the snowballing (SB) approach (Wohlin, 2014). The SB process is executed in iterations starting from a baseline set of papers using Backward snowballing (BSB) and Forward Snowballing (FSB) involves looking through the reference list of the baseline papers and FSB searches for studies that cite papers from the baseline set. The snowballing process comes to an end once no new papers are found on the relevant topic (Felizardo et al., 2016). SB is an efficient and reliable way to conduct a systematic literature review, providing a robust alternative to mechanically searching individual databases for given topics (Wohlin, 2014). The step-by-step process illustrated by Wohlin was followed, in which he affirms that the SB approach finds papers almost similar to the ones in the conventional style of systematic literature review, which involves database search. The SB literature review methodology specified the research question, search strategy, inclusion, exclusion, and quality criteria, and data synthesis. The search strategy, and inclusion, exclusion and quality criteria, are applied as

a part of the SB procedure, as explained in the next section. Execution of the SB procedure was followed by data synthesis. The overview of the literature review methodology is visualised in Figure 2.3.

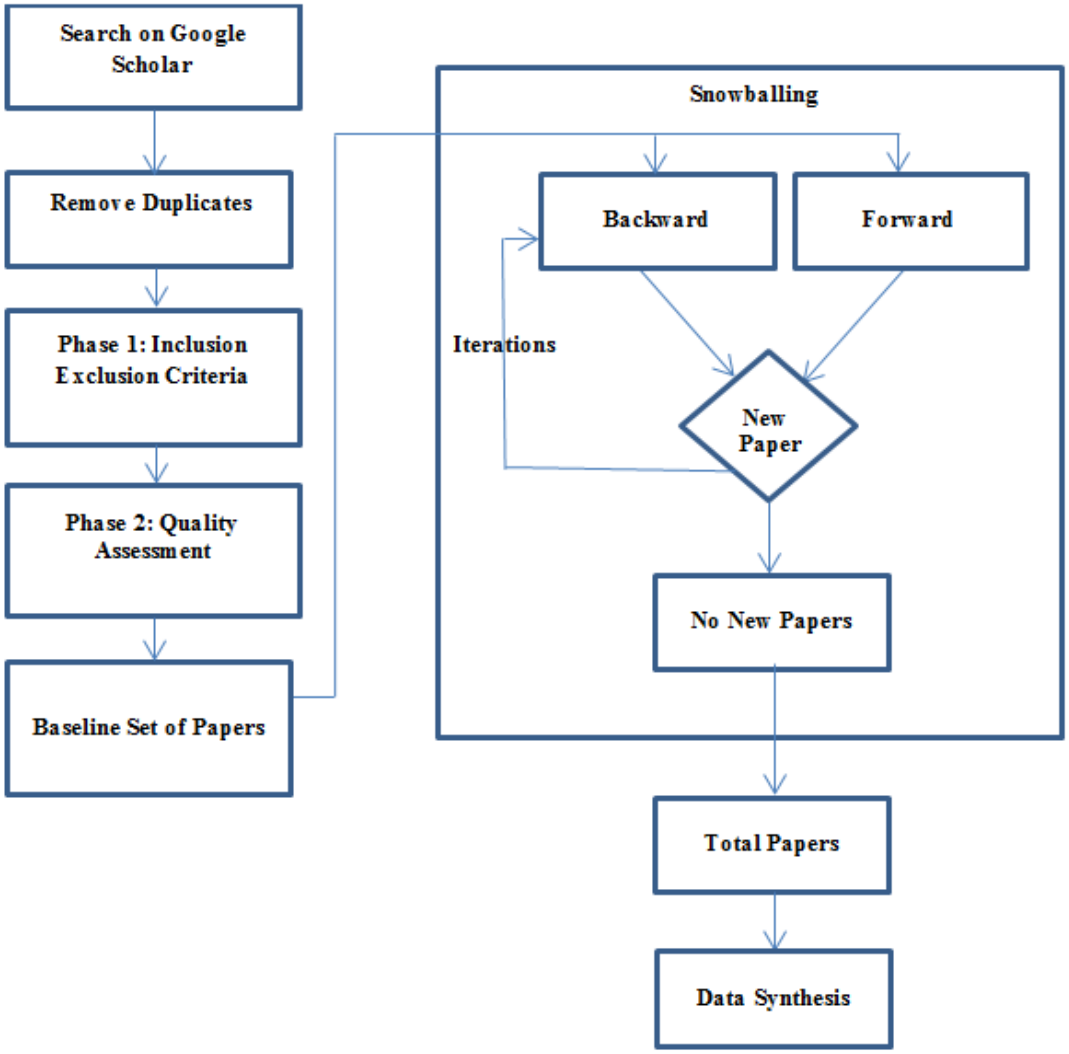

Figure 2.3: Overview of literature review using snowballing

2.3.1 Details on Snowballing Procedure

This section details the SB procedure to search relevant papers. The selection of a baseline set of papers and iterations in SB will rigorously follow the given criteria for inclusion, exclusion, and quality assessment. A detailed view of the SB procedure is depicted in Figure 2.4. The initial baseline papers are pooled from search engine using search string(s) designed to reflect the topic. The final

baseline papers are selected from a pool of initial baseline papers, which are then subjected to subsequent iterations using backward and FSB. Papers found in an iteration are assessed based on relevancy to the topic using inclusion, exclusion and quality criteria. Consequently, new papers are added to the set of final baseline papers. The iterations continue until no new papers are found.

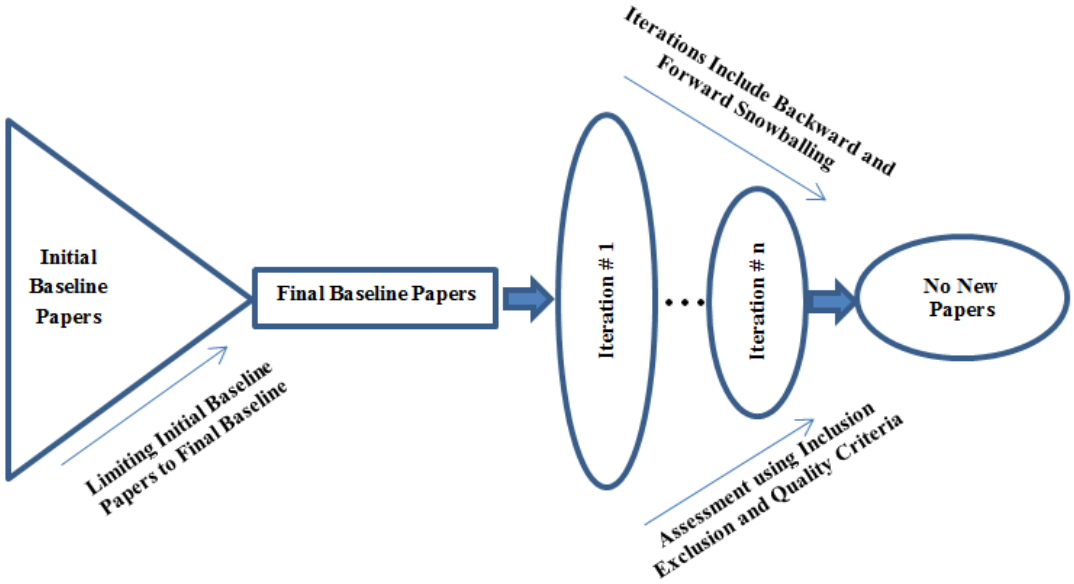

Figure 2.4: The detailed view of snowballing procedure to search papers

2.3.2 Initial Baseline Set of papers

For the purpose of this search, no distinction is drawn concerning the following terms: Free/Libre Open Source Software (FLOSS), OSS, Free Open Source Software (FOSS), and Free Software. The search strings are designed using a combination of all three terms to represent OSS. Google Scholar is used to search each string to avoid bias in the selection of publishers (Wohlin, 2014). Table 2.1 shows the five strings designed to be executed on Google Scholar, to extract an initial baseline set in SB.

Table 2.1: Five search strings to extract initial baseline set in SB

Open source software and turnover and "knowledge loss"
Free open source software and turnover and "knowledge loss"
Free software and turnover and "knowledge loss"
Free/ Libre software and turnover and "knowledge loss"
Libre software and turnover and "knowledge loss"

The SB approach outlined later in this section represents a comprehensive pursuit of related works that are identified in the initial baseline of relevant research papers. Accordingly, this methodology will consider the top 30 search results, provided by Google Scholar for each of the search strings. Most likely Google Scholar search returns the best results in the first few pages. To establish the baseline set of papers it was expected to collect 150 papers from Google Scholar using five search strings and then subject them to systematic inclusion/ exclusion analysis, followed by extensive SB iterations.

2.3.3 Inclusion/ Exclusion Criteria for Baseline Set

The initial baseline set of papers will be further assessed, to be included in the final baseline set, by applying inclusion or exclusion criteria in two phases. In the first phase, papers will be included based on the following criteria:

- The papers that are available in full-text [1]
- Papers written in English
- Papers are peer-reviewed
- Papers relevant to the topic are to be included after reading the title
- When it is unclear from the title abstract and introduction it will be read

[1]The reason for the unavailability of the paper may be because it is simply not published through any of the leading and most widely read outlets (including posting by the researcher themselves online). It should be noted that the researcher had access to all leading research databases.

- When it is unclear from the title abstract and introduction it will be read

In the second phase, the full texts of papers is evaluated utilising quality assessment criteria to make a final decision on inclusion or exclusion of papers into the final baseline set.

2.4 Quality Assessment

The three main concerns relevant to quality assessment are rigour, credibility and relevance as defined below (Dybå and Dingsøyr, 2008):

- Rigour - Has a thorough and appropriate approach
- Credibility - Are the findings well-presented and meaningful?
- Relevance - How useful are the findings to the software industry and the research community?

The above three concerns are assessed through eleven recommended quality assessment criteria in (Dybå and Dingsøyr, 2008) has been effectively used by Kitchenham and Brereton with some adaptions (Kitchenham and Brereton, 2013). In this literature review, the main intention is to find literature relevant to KL due to contributor turnover in OSS. After reviewing the quality criteria presented by (Dybå and Dingsøyr, 2008; Kitchenham and Brereton, 2013), the following check list was formulated to assess papers for their quality assessment, as given in Table 2.2.

Table 2.2: Quality assessment criteria to evaluate papers

No.	Criteria Details
1	Is there a clear statement of the aims of the study?
2	What is the theory or context of research?
3	What research questions are asked?
4	What kind of paper is it? (Problem identification and/or problem solution or Experience Paper, Opinion Survey or Discussion paper)
5	What kind of research was conducted? Experiment, Quasi-Experiment, Lessons learnt, Case study, Opinion Survey, Tertiary Study, Other (specify)?
6	Was the research method appropriate to address the aims of the research? Yes/Partly/No/ Not applicable (i.e. Expert Opinion)
7	Was experimental material or context (for lessons learnt) appropriate to the aims of the research? Yes/Partly/No/Not applicable (i.e. Expert Opinion)
8	For empirical studies (apart from Lessons Learnt), was the data collected in a way that addressed the research issue? Yes/Partly/No/Not applicable (i.e. Lessons learnt or Expert opinion).
9	For empirical studies (apart from Lessons Learnt), was the data analysis sufficiently rigorous? Yes/Partly/No/Not applicable
10	Is there a clear statement of findings? Yes/Partly/No
11	Are there any validity threats and limitations presented? Yes/Partly/No
12	Is the study of value for research or practice? Yes/Partly/No

The given quality criteria will be applied thoroughly during the SB procedure including the selection of the baseline set of papers and subsequent iterations of SB.

2.5 Iterations in Snowballing

Iterations in SB include BSB and FSB described in this section.

2.5.1 Backward Snowballing

In BSB, the references from each paper in the baseline set will be reviewed based upon the relevance to the topic and the inclusion and exclusion criteria will be applied in two different phases similar to the one applied to the baseline set of papers. The only difference will be in inclusion and exclusion criteria for the first phase. The following criteria will be applied in the first phase of BSB (Badampudi et al., 2015):

- Read the title of the referenced paper
- Read the context in which the reference appears in the paper
- Review the abstract of the referenced paper

It is important to understand the context in which the reference is cited within the text of the paper. The referenced paper in full will be assessed based on the criteria of inclusion and exclusion, and quality assessment given in section 2.3.3 and section 2.4 respectively.

2.5.2 Forward Snowballing

Identification of a new paper in FSB is based on the papers citing the paper under examination (Wohlin, 2014). This information is available from the Google Scholar search engine. The inclusion and exclusion criteria is based on the following steps (Badampudi et al., 2015):

- Read the title of the citing paper
- Read the abstract of the paper citing the paper under examination

- Understand the context in which the paper is being cited

The final two steps for FSB are in reverse order as compared to BSB (Badampudi et al., 2015). Reading the abstract before the context of use in FSB will give the researcher clarity on the topic of the paper and enable him or her to understand the context in a better way. The newly identified papers at the end of an iteration in the SB procedure will be added to the final baseline set. The iterations will cease when no new papers are found.

2.6 Application of Snowballing for Literature Review

2.6.1 Identifying the Baseline Set

The queries in Table 2.1 were executed on Google Scholar using Zotero[2] (a free tool for managing bibliographies) to retain Google Scholar results generated by executing the queries and removing duplicates. Zotero further facilitated to download the citation details of the papers including electronic versions. Quotes around the term knowledge loss help to search it on Google Scholar as one word. During the execution of the search strings, three search strings returned results less than 30. For those search strings that returned more than 30 results on Google Scholar, only the top 30 results were selected. In all, 129 publications were extracted and after identifying 62 duplicates 67 individual papers remained. Inclusion/ exclusion criteria given in section 2.3.3, were applied on these 67 publications.

2.6.2 Final Baseline Set

The exclusion criteria were applied in two phases, the outcome of phase one resulted in 21 papers out of 67 papers. One replication study was found (Nassif

[2]http://www.zotero.org/

and Robillard, 2017) based on earlier work in the paper (Rigby et al., 2016) which was not peer reviewed and used the same experimental method. According to the guidelines given when a study uses the same experimental method but different material, it is treated as a multi-case case study, and it is believed to increase the scope and size of the study and not the quality (Kitchenham and Brereton, 2013). The replication study found among 67 papers was excluded. Peer reviewed doctoral symposiums were not included in the initial baseline set, as these short papers (1-4 pages) proposed future work. In case, it was not clear from the title of the paper to include or exclude it, the abstract, introduction and conclusion was read. A number of papers were excluded because they did not discuss knowledge loss or turnover in the context of OSS projects. Furthermore, papers with a focus on knowledge relevant issues in organisations other than in OSS projects were excluded, giving in all 21 papers at the end of phase one.

In phase two, the full text of 21 papers was evaluated based on the criteria to assess quality (section 2.4) and details were retained in an excel spread sheet for future reference. One paper that is on knowledge transfer challenges and mitigation in Global Software Development (GSD) was included, since OSS is considered to be an example of GSD (Herraiz et al., 2006; Lin et al., 2017). Five papers excluded focused on topics including knowledge gaps in post-merger integration, managing knowledge and organisational costs, strategies to manage KL in organisations, a peer reviewed short paper on impact of mentoring with a research model, and on open innovation. The final baseline set consisted of 16 papers that discuss OSS in relation to KL or turnover. The selection of a final baseline set of papers is shown in Figure 2.5. In the next step, 16 papers were iterated as detailed in the following section.

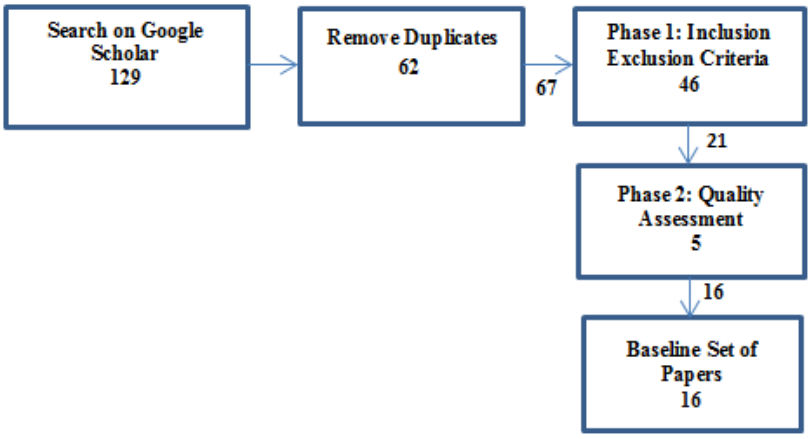

Figure 2.5: Overview of literature review using snowballing

2.7 Iterations

2.7.1 Iteration 1 - Backward and Forward Snowballing

During the first iteration, the first step in BSB was to read titles of referenced papers in the baseline set. In all 643 titles referenced in the baseline set of papers were read for relevance to this work. As a result, 41 papers were identified for further evaluation. It was found that 20 of the papers were already included in the baseline set or the first iteration of BSB. Examination of the remaining 21 papers led to the exclusion of four papers leaving 17 papers to add to the collection of relevant literature. Forward snowballing was applied to 16 papers in the baseline set. The Google Scholar search engine was useful to find a total number of papers citing the corresponding paper in the baseline set. In all, 203 papers cited 16 papers in the baseline set. Only 24 papers were relevant to this work and 23 were already included in either the baseline set of papers or through the first iteration of BSB. Iteration 1 resulted in 18 new papers from the initial 16 papers in the start set.

2.7.2 Iteration 2 - Backward and Forward Snowballing

The 18 papers resulting from iteration 1 were iterated by following the steps of BSB and then FSB. During BSB, 845 references appeared in 18 papers and after reading all titles and applying inclusion exclusion criteria on selected ones, only two new papers were included. Google Scholar was then employed to find out papers that cited the 18 papers from Iteration 1. The papers that were cited in more than 100 publications, titles only for the first 100 papers were read. During FSB, 1174 paper titles were read and two new papers were found. Iteration 2 ended with four new papers.

2.7.3 Iteration 3 - Backward and Forward Snowballing

During iteration 3, four papers resulting from iteration 2 were iterated. In BSB, 252 titles were read for their relevance to this research and no new papers were found. In FSB, 226 papers cited four papers on Google Scholar. However, no new papers were found in iteration 3. Consequently, iteration 3 marked an end to the SB procedure. The SB phase ended with 38 papers, which are listed under primary studies (e.g. PS1, PS2, PS3 and so on) in appendix A.1. The application of the SB procedure is summarised in Figure 2.6.

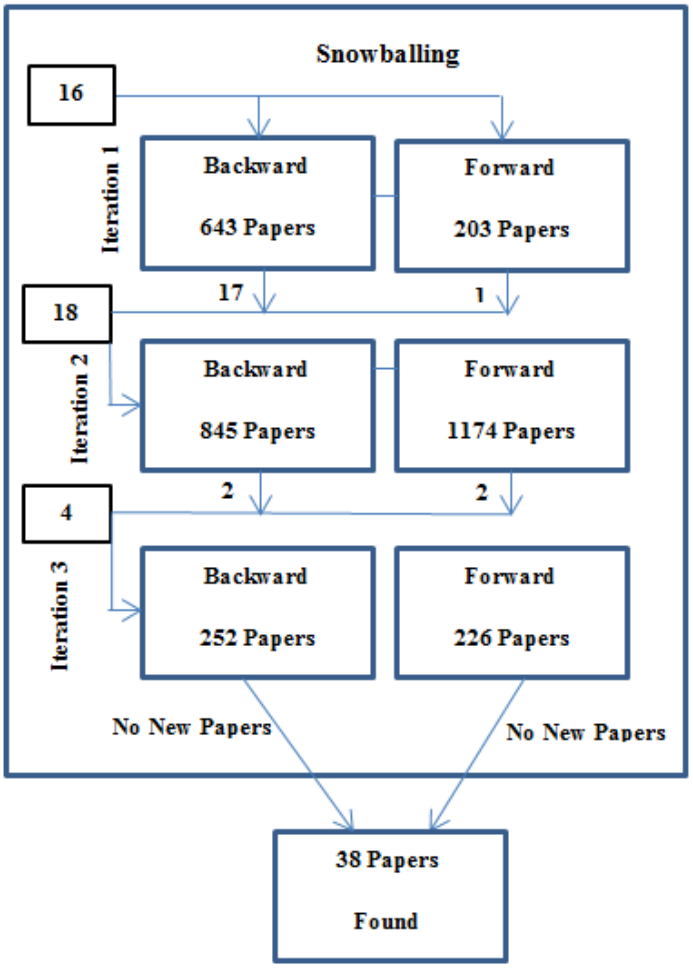

Figure 2.6: Applying snowballing process to search relevant papers on Knowledge Loss in OSS

2.8 Data Synthesis

Data synthesis was performed by collating and summarising the contents of the included 38 papers, similar to the guidelines provided for systematic literature review (Keele et al., 2007). At first, the text of each study is analysed to derive individual themes. Secondly, the derived set of themes are analysed on their relationship with each other based on the inferences made from the study.

The three iterations from the application of the SB procedure yielded 38

papers. During iterations, the general information collected was about paper title, venue of publication, and year of publication. The specific information on papers is constituted upon 12 quality assessment criteria given in section 2.4. The papers selected are from year 2000 to year 2017, the year wise distribution is given in Figure 2.7.

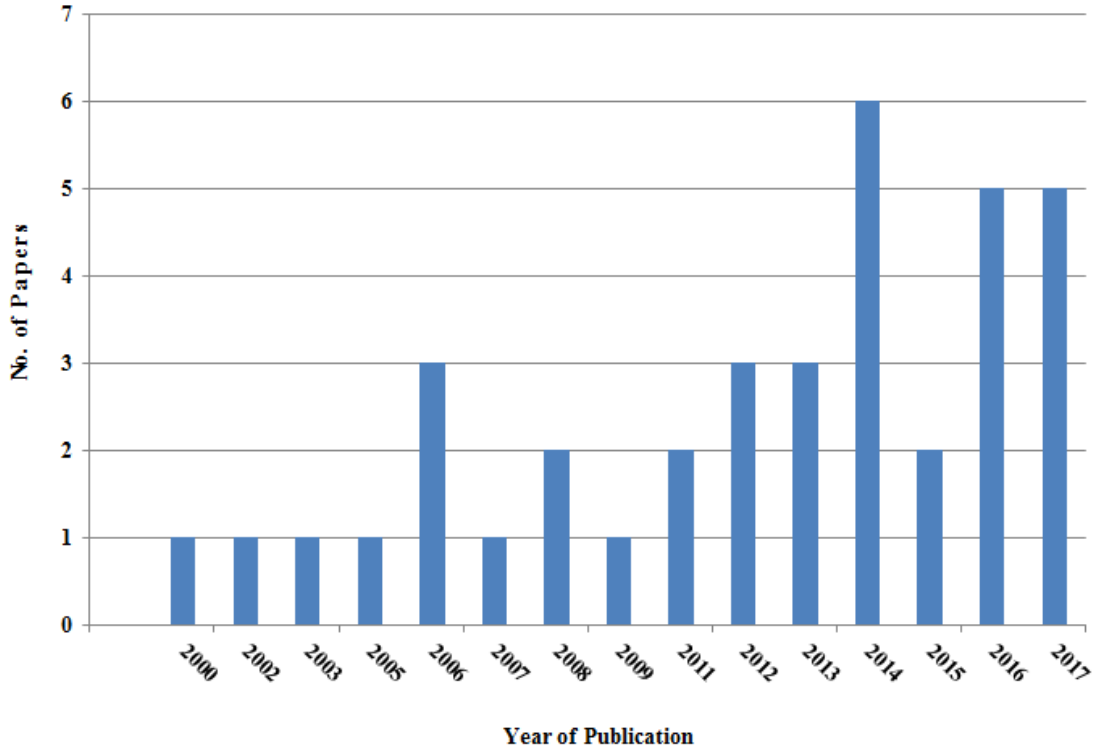

Figure 2.7: The yearly distribution of papers

There are 18 conference papers, 17 journal papers, and 3 peer reviewed papers published in workshops. The distribution of papers based on the research type is shown in Figure 2.8.

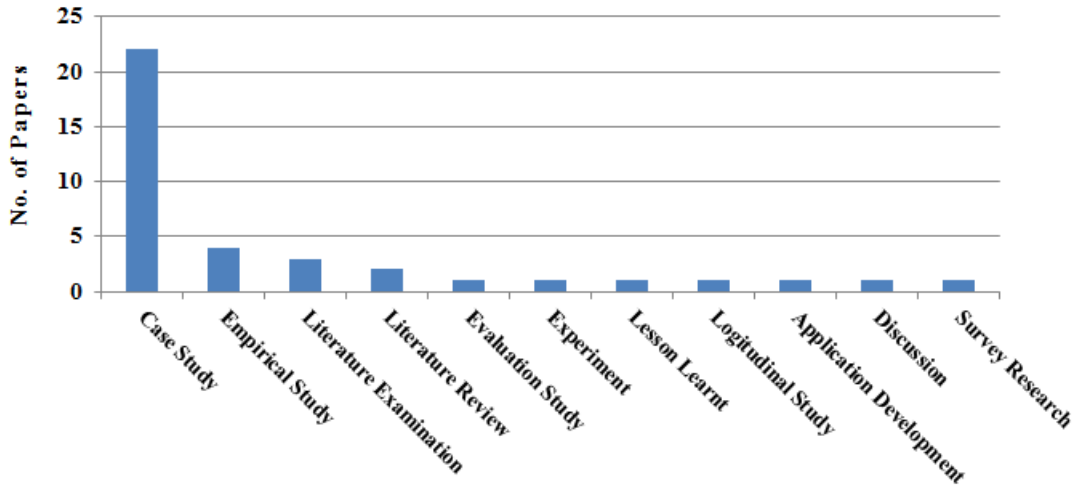

Figure 2.8: The distribution of papers according to the type of research conducted

2.9 Discussion

The emerging themes identified from the papers in this work are subjective in nature and a rigorous process was followed to collect papers in a systematic literature review. In this section, the problem of KL in OSS projects is examined and the subsequent section lists the impact of KL in OSS projects.

2.9.1 Examining Knowledge Loss in OSS

The evolution of an OSS project results in teams of contributors who are constantly joining, leaving, or changing their role in the project. The phenomenon of resources joining and leaving in this fashion is referred to as turnover (Foucault et al. 2015). Some of the reasons for the contributors leaving OSS projects include loss of interest in the project, new career opportunities, no new learning or improvement of skills (Shah 2006). The author of the source code has a strong relationship with the authored code. When the author of the code leaves the project and their code is abandoned, software development can halt due to KL (Rigby et al., 2016). KL is a problem that has been reported equally in CSS (Izquierdo-Cortazar et al., 2009; Jennex and Durcikova, 2013;

Viana et al., 2015) and OSS (Izquierdo-Cortazar et al., 2009; Rigby et al., 2016) projects. The phenomenon of contributors joining and leaving at their discretion is more common in OSS projects than with hired employees in CSS (Robles et al., 2005).

A moderate amount of turnover is thought to bring innovation to the project (Ling et al., 2005) (Ling et al. 2005). Similarly it is reported that in Wikipedia, an online community, moderate levels of membership turnover positively affect collaborative success (Ransbotham and Kane, 2011). However, in OSS projects, contributor turnover is reported to have a negative impact on quality (Foucault et al. 2015) and productivity, as due to KL, extra time is spent to learn the workings of the project (Izquierdo-Cortazar et al., 2009). In many large OSS projects, a high turnover has been observed leading to the formation of the succeeding development teams (Robles and Gonzalez-Barahona, 2006). In order to continue with the software development tasks, succeeding development teams require knowledge about the developed source code. Constant evolution requires that new contributors be knowledgeable enough to perform maintenance tasks on the project during the absence of the earlier owner of the system (Rigby et al., 2016).

A study on the GNOME [3] project reported that 30 months' time is needed for the contributor to understand the software code and to make a contribution (Herraiz et al., 2006). A lack of understanding of the design of abandoned code (Mockus, 2010) slows down the contributors and impacts their productivity (Schilling et al., 2011). Searching knowledge is argued to be time consuming and costly (Von Krogh et al., 2005). The search efforts can vary depending on the source and the level of details. For instance, a post or a query on the project mailing list requires less effort, while searching through the results of search engine use or examining the clues in source code documentation is time consuming (Von Krogh et al., 2005).

[3]GNOME is a well-known large Libre project sponsored by several companies. https://www.gnome.org/foundation/

Accumulated knowledge, which is not codified, is lost once the contributors leave the project. Contributors gain skills, experience while working on software projects, and take the knowledge with them once they leave the project. In this work knowledge, loss refers to the loss of accumulated knowledge that is not available or is too costly to reacquire once the contributor leaves the OSS project.

2.9.2 Impact of Knowledge Loss in OSS projects

In this section, the effects of the KL phenomenon on OSS projects are unfolded and consolidated by synthesis of the data collected from the relevant literature.

A) Abandoned code - When a contributor who has owned or worked on code files leaves the OSS project, the files or Lines of Code (LOC) are orphaned or abandoned (Izquierdo-Cortazar et al., 2009). Abandoned code characterises the situation of KL in terms of efforts lost that affect the productivity of the OSS project. It is emphasised that with the increase of orphaned lines of code, the project may become invisible to the current contributors, since the major parts of the code is written by the contributors who are no longer available (Izquierdo-Cortazar et al., 2009). Researchers have quantified KL based on the number of lines of code and files that become orphaned or abandoned when a contributor leaves (Izquierdo-Cortazar et al., 2009; Rigby et al., 2016). During the preparation of a release, contributors make changes to align their work with the goals of the release (Michlmayr, 2007a). A central person or body then selects a subset of the developed code for the "official" releases and makes it widely available for distribution (Mockus et al., 2002). At the time of release, the unmaintained source code is removed from the project since the original contributors who maintained it are not there anymore. The code has an element of uncertainty for the development team since the contributors who wrote it have left (Izquierdo-Cortazar et al., 2009). Removal of unmaintained code results in loss of existing functionality and may impact users of the system Michlmayr (2007a). It is reported that on average, 12% of Source Line of Code

(SLOC) is abandoned and as abandoned code increases on the project, the numbers of reported defects have been shown to increase as well (Otte et al., 2008). The maintenance of abandoned code is difficult because the team lacks knowledge of its creation and structure (Donadelli, 2015).

B) Project Instability - The stability of the OSS project and its success is dependent on the ability to sustain contributors and a strong combination of experience and knowledge is associated with contributor retention (Schilling et al., 2011). Contributors who work on modifying files by others have a better chance of survival (Lin et al., 2017), since it diversifies their experience and knowledge about the work of others. There is a focus on identifying newcomers at an earlier stage of joining an OSS project, with a higher chance of contributors staying for the long term by being trained by the current contributors Schilling et al. (2012). In another study for the purpose of decision making and observation, the assessment of community stability is based on the estimations of the future participation (Rastogi and Sureka, 2014). Lack of sustained contributors lead to the failure of 80% of OSS initiatives (Colazo and Fang, 2009).

C) Discontinued Knowledge Building - Knowledge building results from collaboration and the use of a variety of tools and resources including Internet Relay Chat (IRC), Concurrent Version System (CVS), bug report, Feature request, patches, tasks and news (Ge et al., 2006). From the initiation of the problem to resolution, contributors with different expertise and knowledge contribute to knowledge building in the community. Knowledge in collaborative online communities is not concentrated with an individual but rather results from the collaboration of contributors. Community members who leave not only take their knowledge and expertise but also the established relationship with other members (Wang and Lantzy, 2011). The knowledge building process may discontinue due to disruption in social ties.

D) Community Health Chaos - The driving force of OSS projects is their communities consisting of ad hoc or informal, foundation or non-profit and

commercial companies (Zheng et al., 2008). Community health is affected by turnover due to loss of human capital and social capital (Wang and Lantzy 2011). Consequently, with human capital the long-term knowledge, which is tacit in nature, is lost. Furthermore, in the case of social capital, the developed relationship among community members is disrupted. Social capital has resources embedded in social relationships with inherent norms and values (Droege and Hoobler, 2003). Furthermore, social structures have a central role in knowledge creation (Nonaka, 1994). Members who are central in the structure of the community, their departure has a higher impact on the community health and membership turnover in Wikipedia, has a significant negative effect on group outcomes, and there exists an association between social capital losses through member turnover and group (Wang and Lantzy, 2011). Moreover, member turnover in Wikipedia is associated negatively with the group productivity (Qin et al., 2014).

E) Damage to the Ecosystem - In a software ecosystem, "collections of software projects are developed and evolve together in the same environment" (Lungu, 2008). Disruption of an ecosystem not only has an impact on the technical aspects (such as its package dependencies), but also on social aspects (Mens, 2016). Ecosystems evolve over time "through socio-technical changes that may greatly impact the ecosystem's sustainability. Social changes like developer turnover may lead to technical degradation" and there is a high probability that contributors will abandon an ecosystem if contributors do not interact in discussion with others (Constantinou and Mens, 2017a). It is reported that when a contributor with a central position in the community unexpectedly left, it was a cause of major disruption in the community (Mens, 2016). More specialised or central leavers cause risk to the functioning of an ecosystem (Lin et al., 2017). For example, a package containing only a few lines of source code but with thousands of dependent projects, when removed, had important consequences "almost breaking the internet" (Mens, 2016).

F) Evolutionary Pressure - When contributors withdraw from a project and new ones join, the transition influences the evolution of the contributor's network and generates evolutionary pressure (Joblin et al., 2017). Evolutionary pressures are due to the difference in turnover rates between core and peripheral developers (Joblin et al., 2017). Evolution of the system will require the new contributor to have knowledge on the project to perform maintenance tasks, in the absence of the owning developers who owned the system earlier. In the absence of a contributor, original owner of the files or system and a successor to files with related knowledge on the project to perform maintenance tasks, the files are at a risk of abandonment (Rigby et al., 2016).

G) Non-uniform Knowledge Distribution - A small subset of contributors, typically 20%, called core members, make major code contributions of about 80% in OSS projects (Mockus et al., 2002). It is worth noting that this distribution is in line with the Pareto Principle (also known as the 80/20 rule) that states for many events, roughly 80% of the effects come from 20% of the causes, demonstrating that most of the activity in OSSprojects is carried out by a small group of people (Goeminne and Mens, 2011). Knowledge distribution in OSS projects among contributors is found to be non-uniform, one developer leaving can cause a major file loss (removal of file ownership leads to its abandonment) in the system (Rigby et al., 2016).It is reported that on the Gimp project, when relevant project knowledge is limited to a small set of contributors, one contributor leaving results in the loss of up to 80% of files in the system and when knowledge is distributed across a larger group of contributors, file loss is minimised when one contributor leaves, as seen in the case of Linux project (Donadelli, 2015).

H) Knowledge Differentiation - Knowledge differentiation refers to the extent of specialisation of team members in different knowledge domains (Chen et al., 2013). Knowledge differentiation in CSS teams shows that knowledge differentiation is specifically important for software development and it involves

integrating knowledge from various domains, such as software architecture, software design methodologies, and business application domain knowledge (Tiwana, 2004). Although OSS teams are different from CSS teams in various ways, the baseline for the knowledge intensive nature of software development does not change for CSS or OSS contributors. In the case of Freenet, a file sharing OSS project, it was found that the majority of contributors specialize in coding few and not all modules, because of existing knowledge barriers between the modules (Von Krogh et al., 2003). Coding specialization exhibits knowledge differentiation in OSS teams (Chen et al., 2013). The knowledge differentiation concept intertwines with knowledge distribution and can have consequences in OSS projects when the contributors leave. The impact of contributor turnover has been realised in the software engineering community and it has been asserted that most existing studies are conducted in the open source communities as compared to non-open source communities. The data collection is mainly from software project management and obtained by performing qualitative observations and quantitative analysis of contributor activity (Bao et al., 2017). This section discussed the main impacts of Knowledge Loss in OSS projects. The impact of KL on quality and productivity was discussed in section 2.9.2. Similarly, the role of knowledge in relation to maintenance, reinvention, and evolution was discussed in section 2.2.1, all of which is illustrated in the mind map in Figure 2.9. The list of primary studies appear in mind map within parenthesis abbreviated as PS1, PS2, PS3 and detailed in appendix A.1

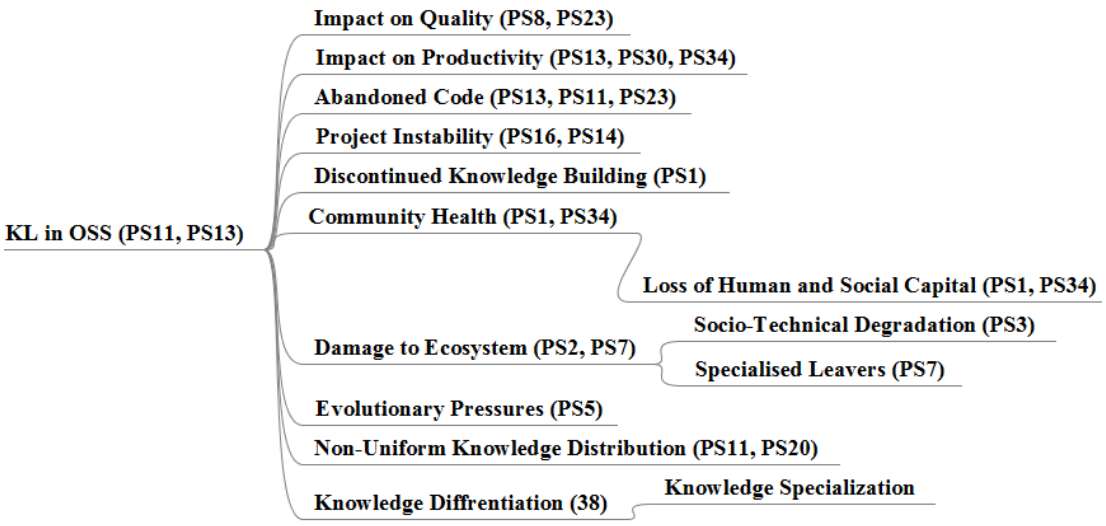

Figure 2.9: Impact of knowledge loss in OSS projects

2.10 Reducing Knowledge Loss in OSS Projects

The KL phenomenon and its impact in OSS projects were discussed in 2.9. In this section, the focus is on the literature that discusses the reduction of KL in OSS projects due to contributor turnover. Attention is first directed to Knowledge Retention (KR) in organisations, which mainly comes into focus when an employee is leaving and the need for a KR mechanism in an organisation is assessed based on the following (Lindvall and Rus, 2003):

1. A lack of knowledge in the organisation and an overly long time to acquire knowledge due to a steep learning curve

2. Employees repeating mistakes and performing rework because they forget what they learned from previous projects.

3. Chances of employees owning key knowledge becoming unavailable

Knowledge Retention relates to capturing knowledge in an organisation and is an important aspect of Knowledge Management (KM). KM is defined as the approach adopted by an organisation to engage workers in relevant activities of creating, managing, sharing and reusing knowledge (Dingsøyr et al., 2009). KM

also refers to coordination and exploitation of knowledge in an organisation and making the right knowledge available to the right people while benefiting from it for competitive advantages (Drucker, 1999). KM enables an organisation to enhance its capabilities to deal with new and unusual situations (Choi et al., 2008). Knowledge retention can be seen as a way of embedding and enabling knowledge within an organisation and a critical factor for sustainable performance (Doan et al., 2011).

The KR strategies in an organisation advance innovation, organisational growth, efficiency, employee development, and competitive advantage. Sometimes techniques such as exit interviews are used to capture the knowledge of the employee. Some organisations also follow formal knowledge retention strategies. It is an effort-demanding task to identify potential knowledge for the organisation. The structure of the organisation in the context of how well it supports knowledge retention is of importance as well. Once the person who has the potential knowledge leaves the organisation, it is hard to retain this knowledge.

Codification and personalisation are considered useful strategies for managing knowledge intensive activities like software development (Donnellan et al., 2005). Codification is the documentation of the knowledge to be stored, disseminated, and reused. Personalisation is the development of networks to connect people where they can share tacit knowledge (Hansen et al., 1999). In knowledge bases, codification captures electronic information and personalisation deals with the ways humans use and process knowledge (Donnellan et al., 2005). Organizations implement codification strategy to encourage the reuse of explicit knowledge.

The core techniques designed to retain knowledge in an organisation are mainly dependent on its knowledge-sharing practices. There are many techniques that facilitate knowledge capture, sharing and reapplication, namely after-action reviews, communities of practice, face-to-face meetings, mentoring programs, expert referral services, video conferencing, interviews, written

reports, use of training and technology-based systems to transfer the knowledge (De Long and Davenport, 2003).

Table 2.3: Representation of knowledge retention techniques with knowledge conversions type and codification strategy

Knowledge Retention Techniques	Knowledge Conversion Type	Managing Strategy
Community of Practice	Tacit to Tacit	Personalisation
Post-mortem	Tacit to Tacit	Personalisation
Mentoring	Tacit to Explicit	Codification
Storytelling	Tacit to Explicit	Codification
Experienced Based Memory	Tacit to Explicit	Codification
Interviews	Tacit to Explicit	Codification
Training	Explicit to Tacit	Personalisation

In Table 2.3, KR techniques practiced in organisations with the type of knowledge conversion and managing strategy are given. Community of Practice[4] are used as a long term strategy by management to retain knowledge before it is lost (De Long and Davenport, 2003), where experienced members deal with the problem of losing expertise by contributing valuable lessons to member companies. Post-mortem reviews are a practical method to capture and reuse experience from projects that are complete or have finished a major activity phase (Dingsøyr, 2005). Mentoring is suggested to be a logical approach for transferring important tacit and implicit knowledge (De Long and Davenport, 2003). Story telling can play a valuable role in transferring knowledge to convert tacit knowledge into explicit knowledge (De Long and Davenport, 2003). Software engineers in their daily work environment, create projects and product memory, which are based on their experience (Rus et al., 2002). Experienced based memory is built on software engineering practices and product memories, which are an indirect or direct effect of software development. Some examples include version control systems, change management, documentation design decisions, and requirements traceability. Interviews in an organisation is a

[4]A community of practice is a group of people who share a concern or a passion for something they do, and learn how to do it better as they interact regularly

knowledge transfer process used to integrate the knowledge captured into the organisation (De Long and Davenport, 2003). Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching and shadowing (De Long and Davenport, 2003).

KR practices that are used in organisations to overcome KL due to leaving employees have been briefly discussed. In contrast to organisations, KR is not formally practiced in OSS projects. The research community recognises KM in OSS development as challenging because of its highly distributed, dynamic work structure, and knowledge-intensive characteristics (Ciborra and Andreu, 2001; Crowston et al., 2012). KM is one of the social processes and is of potential importance to manage the knowledge necessary for a successful development effort in OSS projects (Crowston et al., 2012). "Social processes capture cognitive, verbal, and behavioural activities performed by team members to manage interpersonal relationships among them" (Marks et al., 2001).

The research on knowledge management in OSS projects has focused on knowledge reuse (Dafermos, 2005; Hemetsberger and Reinhardt, 2004; Huysman and Lin, 2005; Lakhani et al., 2003; Lee and Cole, 2003; Singh et al., 2006; Von Krogh et al., 2005), cross boundary learning in source communities (Huysman and Lin, 2005), knowledge sharing involving strategic interaction (Kuk, 2006), learning theory on knowledge creation and sharing in online communities (Lee and Cole, 2003), and learning driven by criticism and error correction and a social view of learning that have overcome problem of tacit knowledge transformation (Hemetsberger and Reinhardt, 2004). The details are summarised in Table 2.4 including the name of the KM related activity, description of study and literature references.

Table 2.4: Summary of Research focus on KM relevant activities in OSS (Crowston et al., 2012)

KM Activities	Description	Literature Reference
Knowledge Reuse	Knowledge shared or reuse in OSS development	Dafermos, 2005; Hemetsberger and Reinhardt, 2004; Huysman and Lin, 2005; Lakhani et al., 2003; Lee and Cole,2003; Singh et al., 2006; von Krogh et al., 2005
Cross Boundary	Online communities without strict membership requirements activate cross-boundary learning and knowledge sharing	Huysman and Lin, 2005
Knowledge Sharing	Knowledge sharing and strategic interaction	Kuk, 2006
Learning Theory on Knowledge Creation and Knowledge Sharing	Learning theory provides an understanding of online knowledge creation and sharing	Lee and Cole, 2003
A Social View of Learning	A social view of learning and knowledge creation and solution to resolve the problem of tacit knowledge transformation through technological tools, task-related features, collective reflection, stories, and usage scenario	Hemetsberger and Reinhardt, 2004

Further research is required to explore suitable KR practices applicable in OSS projects as indicated by one of the questions raised on mechanisms and team norms that are used to store knowledge contributed by team members. In CSS organisations KR mainly comes into focus when an employee is leaving an organisation (Lindvall and Rus, 2003). On the contrary, in OSS projects the unpredictable nature of commitment from contributors creates an element of risk (Robles and Gonzalez-Barahona, 2006). In OSS, contributors can leave since they are not under any contractual binding as in CSS organisations.

In this literature review, it was found that KM relevant activities of knowledge creation and knowledge sharing are evident in OSS projects as

discussed in the following section 2.11. Furthermore, literature examination directed the researcher to 10 mitigations to reduce the impact of KL due to contributor turnover in OSS projects detailed in the following section 2.12.

2.11 Manifestation of Knowledge in OSS Projects

OSS projects are considered an example of GSD (Herraiz et al., 2006). OSS contributors are scattered globally while working together on projects. They collaborate on tasks through asynchronous technology mediated channels and utilise them to acquire and share knowledge. Earlier knowledge in this work is defined as "experience and expertise built by contributors that evolves from the day-to-day interaction on the OSS project'". The manifestation of knowledge in OSS projects is experienced in operations of OSS communities. The knowledge manifestation in OSS projects can be explained by three categories consisting of an object of knowledge development, a process of knowledge development, and a location of knowledge (Venzin et al., 1998). Accordingly, the object of knowledge refers to procedural knowledge and knowledge of events including trends within and outside the organisation. Further, the knowledge development process refers to either cognitive abilities or the knowledge construction process. The knowledge construction process includes knowledge creation, sharing, transfer, and its application. Finally, location of knowledge refers to carriers of knowledge, who can be individuals, groups, organisations, inter-organisations, and customers. Principally the location of knowledge is an indication towards tacit knowledge, which requires physical presence. Availability of tacit knowledge is dependent on its transfer and encoding before the contributor leaves the organisation. Contributors who participate in OSS projects with various forms of contributions (Emanuel, 2014) have project relevant tacit knowledge and are responsible for its transfer and encoding before they leave the project.

No specific details on Knowledge Transfer (KT) in OSS projects was identified. Since OSS projects are considered an extreme case of GSD, a paper found earlier was revisited, and through the SB process that investigated the mitigation strategy on KT in GSD settings (Nidhra et al., 2013). The paper enlists mitigation strategies based on personnel, project and technology factors. Some mitigation strategies similar to the ones listed in section 2.12 were identified. In order to resolve personnel challenges mitigation strategies applied are mentoring and shadowing, proactive learning and peer-to-peer help. In addition, project relevant challenges are mitigated by acquiring knowledge from community of practice and by maintaining documents and process. Finally, technological challenges are overcome by using document management systems, e-mails, wikis, instant messaging, configuration management system web-based mentoring, and access to a knowledge repository.

The following sections discuss knowledge creation and knowledge sharing as part of the knowledge construction process in OSS projects, based around literature found in this review.

2.11.1 Knowledge Creation

Knowledge creation takes place when individuals are collectively working and interacting on a task and are constantly acquiring relevant knowledge. Nonaka et al. (Nonaka et al., 2000), explain the knowledge creation process involving conversion of tacit knowledge to explicit knowledge which is then "crystalized". The process of knowledge creation is based on four modes of knowledge conversion: Socialisation, Externalisation, Internalisation, and Combination are coined as SECI. Socialisation is the sharing of experience and results in the creation of new tacit knowledge from the existing tacit knowledge. Externalisation is the conversion of tacit knowledge to explicit knowledge. Externalisation results in articulated knowledge. Combination is the addition of the new explicit knowledge to the existing explicit knowledge in the knowledge

system. Internalisation is the conversion of explicit knowledge to tacit knowledge. In internalisation, knowledge is acquired from artifacts in explicit form, and new mental models are created resulting in tacit knowledge. The process of knowledge creation as detailed by SECI, can be used to explore knowledge creation in OSS projects (Rashid et al. 2017).

Further, two dimensions of knowledge are said to be widely used namely tacit vs. explicit and individual vs. collective (Nidhra et al., 2013). Individuals create individual knowledge and it exists with them, while collective actions of group leads to the creation of collective knowledge (Nonaka, 1994). Furthermore, founded on the combinations of the two dimensions of knowledge, four types of knowledge are proposed (Lam, 2000):

- Embrained Knowledge: Individual - Explicit (e.g. theoretical knowledge)
- Embodied Knowledge: Individual - Tacit (e.g. practical experience)
- Encoded Knowledge: Collective - Explicit (e.g. written rules, procedures)
- Embedded Knowledge: Collective - Tacit (e.g. routines, norms)

Knowledge creation in OSS projects differs from the CSS organisation, as highlighted through five organising principles of the community based model: intellectual property ownership, membership restrictions, authority and incentives, knowledge distribution across organisational and geographical boundaries and dominant mode of communication (Lee and Cole, 2003). In CSS organisations, the knowledge is owned by the organisation with access given to employees, the knowledge distribution is within the boundaries of the firm and mostly with face-to-face communication. On the contrary, in OSS projects, the knowledge contributions and sharing is open without any membership constraints, distribution of knowledge extends outside the community, and interaction of contributors is on a larger scale than in CSS organisations. In OSS projects, knowledge creation is through social interaction among

individuals and organisations, and it is dynamic in nature (Nonaka and Takeuchi, 1995).

2.11.2 Knowledge Sharing

In OSS projects, knowledge sharing is an ongoing activity in an intensely people-oriented and self-organised community (Sowe et al., 2008). Knowledge sharing is declared as a cognitive task and an OSS team is considered an example of a complex cognitive system (Chen et al., 2013). Knowledge sharing is through asynchronous means of communication and with a collection of artifacts, which are publicly available for reuse (Rashid et al., 2017). OSS contributors demonstrate their technical competence by actively participating in mailing-list discussions, reporting bugs and submitting code (Von Krogh et al., 2003). In a longitudinal study, it was established that core contributors' knowledge sharing behaviours brings a high level of skills and understanding to the project teams (Licorish and MacDonell, 2014). Effective knowledge sharing improves the productivity of the project (Levy and Hazzan, 2009).

Measures of knowledge sharing activities identified with knowledge posting and viewing are indicators of the good state of health in virtual communities (Koh and Kim, 2004). Utilising a similar concept of knowledge sharing activities such as posting and viewing, the Knowledge Sharing Model (KSM) in Figure 2.10 is developed, by analysing e-mail exchanges in Debian projects among a list of participants (Sowe et al., 2008). The KSM model follows a constructivist approach, which stresses the active and autonomous role of the learner and knowledge is constructed based on the learner's interaction with the learning environment while prior knowledge impacts the learning process (Sowe et al., 2008).

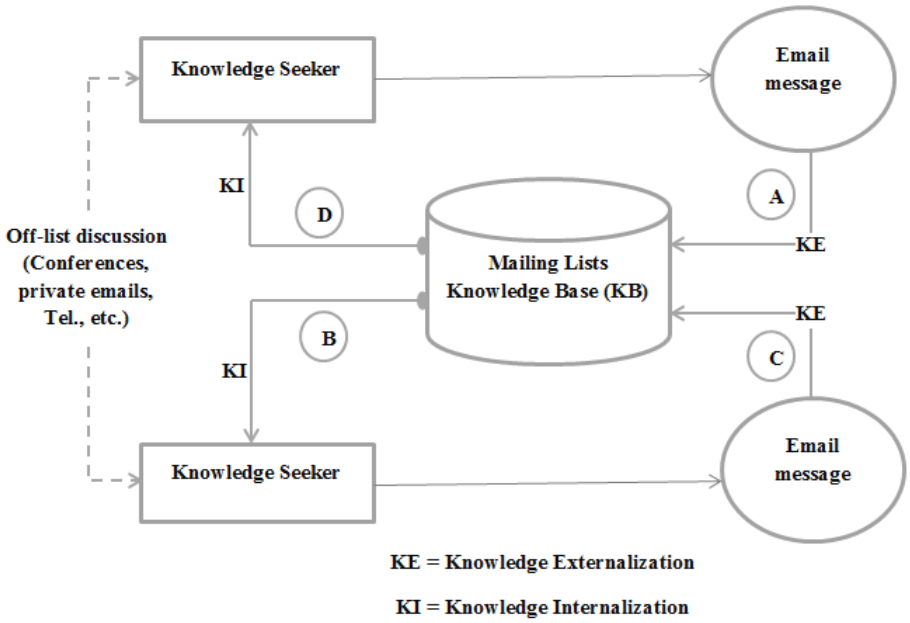

Figure 2.10: Knowledge Sharing Model (KSM) (Sowe et al., 2008)

The model uses a simple measure of two values consisting of e-mail messages posted by knowledge seekers and the number of replies posted by knowledge providers. The analyses of knowledge sharing trends and information contained in e-mails can be utilised to identify what kind of knowledge exchanges are happening in the contributors' community. Moreover, the number of contributions made on the project can determine the expertise level of the group responding to knowledge seekers.

A study on the schema of information types sought in OSS mailing lists asserted that mailing lists are a strong representative of communication in OSSand offer an insight into information seeking needs (Sharif et al., 2015). The findings suggest that 42 percent of information sought on mailing lists is on understanding task implementation and understanding bugs. Mailing lists are a primary source of communication in OSS where knowledge sharing is abundant (Sowe et al., 2008). Programmers never meet face to face but coordinate all their activities (Mockus et al., 2002). In an OSS project, a mailing list is reported to have a multidimensional construct including knowledge differentiation, knowledge location and knowledge credibility (Chen et al. 2013).

Knowledge differentiation refers to the extent of specialisation in different knowledge domains, knowledge location is about who knows what, and knowledge credibility is assessed by the task performance of individual developers.

Gamification (Yilmaz et al., 2016) is another emerging form of knowledge sharing in OSS communities (Vasilescu et al., 2014). The community members vote upon the questions and answers posted on a site, the numbers of votes reflect the poster's reputation and a measure of his or her expertise by the potential employer. A gamification element on sites is found to have increased the engagement of the participants and popularity of the site. In OSS communities, gamification is argued to provide a better visibility of contributors activities (Vasilescu et al., 2014).

The social media sites also serve for contributors to learn, collaborate, share knowledge and interact with users of software (Vasilescu et al., 2014). Contributors contribute on software development sites such as GitHub for coding, Jira to track issues, StackExchange network for open query submission and response, StackOverflow for professional programmers and CrossValidated for statisticians and data miners. Blogs are also considered an effective way of sharing knowledge due to easier access to the internet (Hsu and Lin 2008). Knowledge is also stored in repositories namely: Concurrent Versions System (CVS), Subversion (SVN), Frequently Asked Questions (FAQs), project websites, blogs, bug reporting and bug tracking databases (e.g. Bug Tracking System BTS) and mailing lists (Sowe et al., 2008). Figure 2.11 summarises primary studies in a mind map on literature that discusses knowledge creation and knowledge sharing in OSS projects.

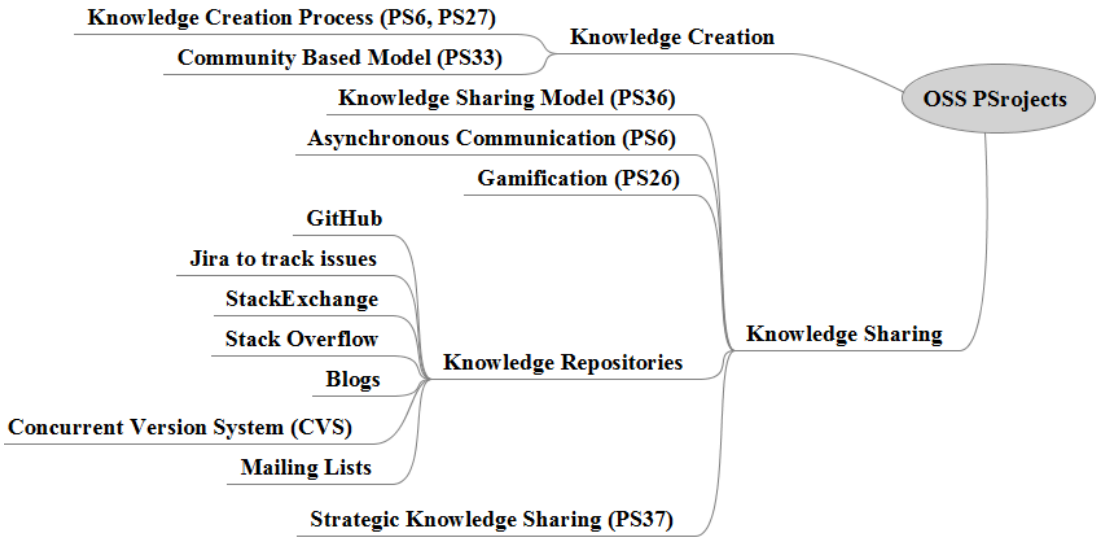

Figure 2.11: Mind map of the knowledge creation and sharing in OSS projects

2.12 Knowledge Retention in OSS Projects

The examination of selected literature on KL in OSS projects resulted in themes including mitigation of KL in OSS projects. A body of researchers from the software engineering community have focused on the impact of turnover in OSS projects and have highlighted some ways to alleviate its effect, referred as mitigation techniques at this point of research. The identified mitigation techniques are discussed in this section.

A) Visualisation of Resources - A visualization word cloud has been proposed to show quickly the level of cooperation of the team in the project (Fronza et al., 2013). A large variety of data collection is without human intervention and rendered as a Wordle. Intensity of colour and size of the letter in a Wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors.

B) Pair Programming and Shared Code Ownership - In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested (Mens, 2016).

C) Successor - A successor is a person who has relevant expertise and is

knowledgeable on the work of other contributors. Identification of successors and involving them as co-owners is presented as a method to reduce the risk associated with developer turnover (Rigby et al., 2016). The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks (Rigby et al., 2016).

D) Centralisation - Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality (Britto et al., 2016). The situation of KL faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams.

E) Removal of Knowledge Barriers - Only a few OSS contributors transit to a higher learning state, due to high learning barriers. Consequently, it can take newcomers up to 60 weeks to become an effective contributor to a OSS project (Adams et al., 2009). Knowledge retention in OSS projects can also be improved by the removal of knowledge barriers: namely, lack of technical experience, lack of domain expertise and lack of project practices that hinder the contributions (Steinmacher et al., 2015a,b).

F) Knowledge Map and TMS - An insight into the area of expertise members, a knowledge map or directory can be used on the project website (Chen et al., 2013). Organizations such as IBM have used such a directory as their internal knowledge portal. In order to leverage OSS project teams, a focus is required towards facilitating the TMS development within the teams, based on knowledge location, the usage of the developer mailing list and knowledge credibility (Chen et al., 2013). These dimensions are reported to have positive effects on communication quality, improving team performance and reduce the impact of turnover.

G) Diversity of Core Contributors - For an OSS project to survive, a diversity of core developers is required (Wahyudin et al., 2007). When a key contributor abandoned an OSS project, it revealed a very fluctuating proportion of developer contribution. A significant imbalance between the contribution and the response from the developers' community was noticed. The reason for the dying project was that a diversity of core contributors were missing from the project (Wahyudin et al., 2007). Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next.

H) Uniform Knowledge Distribution: - The communication of the OSS project is directed by the core contributors. Their attitudes and involvement in knowledge sharing were linked to the demands of their wider project teams (Licorish and MacDonell, 2014). The core contributors bring high levels of skills and cognitive characteristics to their project teams. They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities. The more code changes, core developers perform, the more knowledge they provide (Licorish and MacDonell, 2014). However, their least involvement in communication and task changes results into some negative team attitudes. This kind of disruption in communication in OSS projects can hinder knowledge sharing. Another resolution to the non-uniform distribution of knowledge may be the proactive assignment of maintenance tasks on the code written by other contributors. As indicated that contributors who modify codes from other contributors stay longer on the project (Lin et al., 2017). This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.

I) Gamification - A gamified environment has important implications for knowledge management in software engineering (Vasilescu et al., 2014) and OSS projects. As observed, Q & A gamification increased the engagement of

knowledge providers and the quickness of response. This finding suggests that Q & A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted (Zagalsky et al., 2016). On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer, gains the most points (Zagalsky et al., 2016). Knowledge is curated in gamification other than being developed, as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clear of what seems to be unnecessary information (Zagalsky et al., 2016).

J) Improving Code Review Feedback Time for non-Cores - Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors (Rigby et al., 2014). As indicated by a social network analysis of the code review data from eight popular OSS projects, core developers as compared to peripheral contributors have the benefit of receiving quicker feedback, face shorter review intervals and have a higher code acceptance rate (Bosu and Carver, 2014). Due to the lack of an established reputation, peripheral developers wait 2 to 19 times (or 12 to 96 hours) longer than core developers to complete the review process. Accordingly, a delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor (Bosu and Carver, 2014). An improvement to the timings of the review feedbacks in OSS projects to peripherals can result in a uniform distribution of knowledge, reduce the effects of turnover, and motivate newcomers to stay for a longer duration.

This section explained the manifestation of knowledge in OSS projects using the knowledge construction process mentioned in section 2.11. It was found that in OSS projects, knowledge creation, and knowledge sharing is abundant through

asynchronous communication. Section 2.12 discussed 10 mitigation techniques to reduce KL in OSS projects visualised as a mind map in Figure 2.12.

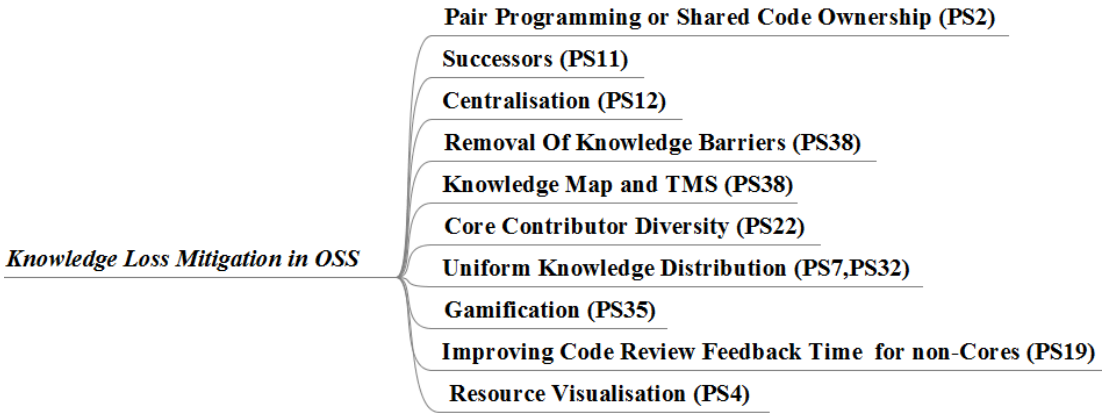

Figure 2.12: Mind map of mitigation approaches to knowledge loss in OSS projects

In the literature relating to OSS projects the emphasis is on the retention of contributors (Constantinou and Mens, 2017a; Schilling et al., 2012), the stability of contributors (Lin et al., 2017; Rastogi and Sureka, 2014; Robles and Gonzalez-Barahona, 2006), reducing the associated risks by identifying successors to the maintenance task (Rigby et al., 2016) and effects of turnover mainly on quality and productivity (Foucault et al., 2015; Izquierdo-Cortazar et al., 2009; Qin et al., 2014; Rigby et al., 2016). One study also reflected on the behaviour of the contributors in OSS projects (Garcia et al., 2013), the presence of negative emotions before contributors leave.

In CSS projects, the contributors with different roles such as developers, analysts, and testers are bound by legal contracts that ascertain that they manage conflicts and demonstrate a certain level of collaboration. In OSS projects, the transient nature of contributors means they may not cooperate well if confronted with conflicts or when they are not motivated (Garcia et al., 2013). Furthermore, in OSS projects, the review found no evidence of specific rules or formal process for the initiation of KR process before a contributor leaves.

Typical strategies for KR in CSS are reactive in nature. These strategies

include knowledge capture and storage by codification, conducting interviews and identifying lessons learned or best practice from projects that departing employees made contributions to (Daghfous et al., 2013). The outcome of these strategies captures a small portion of the knowledge created, acquired and used in the workplace (Daghfous et al., 2013). Similar to CSS, the knowledge sharing in OSS projects is reactive. The contributors ask for information when required through the asynchronous means of communication. In OSS projects, proactive knowledge retention practices are required that resonate with the work environment and are non-intrusive, non-invasive, without any overhead on the productivity of contributors, and promise freedom to practise by free will.

While elaborating knowledge collaborations in social media communities, emphasis is on two stages, the creation stage when information is developed and formed, and the retention stage when the created information is preserved and refined through ongoing collaboration (Ransbotham and Kane, 2011). Contributor turnover is ongoing even after the community has a successful collaboration, it is suggested that social media communities must create and retain knowledge (Ransbotham and Kane, 2011). In the case of OSS communities, it can be implied that KR is also a result of a two-stage process, where the first stage is about knowledge creation and the second stage enables the retention practices to manage knowledge.

The knowledge gap created due to turnover of contributors in OSS projects indicates missing KR practices. In Figure 2.12, ten mitigation techniques to reduce KL are highlighted. Nine of these mitigation techniques focus on the practices that can be followed in OSS projects. For instance, improving the timing of feedback would be beneficial to boost the intrinsic motivation of non-core contributors and achieve uniform knowledge distribution on projects. The knowledge management literature emphasizes intrinsic motivation in promoting employees' knowledge sharing due to its consistently positive and lasting effect (Pee and Lee, 2015). The practice of gamification is another way

to share knowledge but also has implications for extrinsic motivation. The prospective employer will choose the contributor with the best answer to a question. Knowledge Management in OSS projects is considered a challenging task. Abundant knowledge sharing was found but not any specific practices for KR in OSS projects. The concept of health metrics is proposed for stakeholders to assess whether a project initiative is likely to be sustainable and is worth supporting (Wahyudin et al., 2007). A status overview of health in OSS projects is evaluated from the project data available on web repositories. KM should be one of the evaluation factors to assess the knowledge sharing activities of OSS projects for their sustainability in the future. The field might benefit from additional metrics for KM evaluation in OSS projects, similar to the ones provided to evaluate the health of OSS projects such as developer and user community liveliness, product quality by using defect management of open issues and service delays, communication through number of downloads mailing lists posts and response rate.

2.13 Chapter Summary

The objective of this literature review was to understand the phenomenon of KL due to contributor turnover in OSS projects. In order to understand the phenomenon of KL in OSS projects, a literature review using SB as a search strategy was employed. The review identified 38 papers after filtering from a large number of papers (more than 2000) in a comprehensive search. The papers spread over the period of the year 2000 to 2017. The majority of the papers employed empirical methods as their research methods. This review identified 10 impacts from KL and 10 mitigation techniques to overcome KL in OSS projects

OSS projects are considered an extreme case of global software engineering with dynamic, dispersed, and transient contributors collaborating through technologically mediated channels. In comparison to contributors in CSS

organisations, contributors in OSS are not contractually bound. To perform maintenance tasks on OSS projects, knowledge is acquired using asynchronous communication where the knowledge seeker asks questions. Delays incurred acquiring certain kinds of knowledge to perform various tasks impacts the productivity of the contributor and the overall project. Further, the lack of knowledge results in poor quality code and increases the number of defects in the code repository.

Organisations invest in KM activities to organise, create, share, reuse, transfer, and retain knowledge. Knowledge sharing was found to be abundant but there was no evidence of knowledge retention to reduce the impact of KL in OSS projects. Moreover, knowledge sharing is reactive in nature, initiated by the contributor while looking for task-relevant knowledge. This suggests that there is insufficient attention paid to KM in general in OSS, in particular, there would appear to be an absence of proactive measures to reduce the potential impact of KL. This is the finding of the robust literature review employed, it has enabled a complete response to Research Question 1, and it justifies further research effort to identify proactive knowledge retention practices for OSS projects.

This chapter identified that phenomenon of KL exists in OSS projects and currently there is no existing mechanism for KR in OSS projects. In the following chapter a research methodology is presented to further investigate the gap identified in this literature review of non-existing KR mechanisms to reduce KL in OSS projects.

Chapter 3

Research Methodology

3.1 Introduction

This chapter discusses the research methodology contributing to the formation of proactive knowledge retention practices in OSS projects to transform contributors' use of knowledge, and engagement in knowledge relevant activities including knowledge sharing and knowledge transfer. The research methodology presented in this chapter is already published in (Rashid et al., 2018) which was extended as an invited journal publication (Rashid et al., 2019a). The text of this chapter largely mirrors the text and content of these previously published artefacts which were completed as part of this research work.

This chapter discusses the research methodology designed to investigate the research question RQ2 detailed in chapter 1 section 1.4 and reiterated here:

> **RQ2.** What are the effective knowledge retention practices in OSS projects?

In order to investigate RQ2, the following two sub-questions need to be answered, which are reiterated as follows:

> **RQ2.1** How can a comprehensive set of knowledge retention practices be developed for OSS projects?

> **RQ2.2** How can effectiveness of knowledge retention practices be evaluated in OSS projects?

The chapter is structured as follows: section 3.2 entails the OSS project structure, section 3.3 describes philosophical concerns in research, section 3.4 discusses different research designs and one for this research including data analysis and validity concerns. Section 3.5 explains the empirical implication of this research. Finally, section 3.6 summarises the chapter while recollecting the important details.

3.2 OSS Project Structure

Contributors in OSS projects interact on technology-mediated channels to acquire and to share knowledge such as mailing lists, forums, and Internet Relay Chat (IRC). The contributors who are skilled and experienced on a specific module of the project may not explicitly share their acquired knowledge and on their leaving, the project suffers the knowledge gap. In contrast to contributors in OSS projects, employees in traditional organisations may be under contractual obligation to notify their employer before leaving the organisation and to fulfil a notice period during which knowledge transfer concerns can be addressed. The workforce in OSS projects is of a transient nature due to inevitable turnover (Michlmayr, 2007b; Xu, 2006; Yu et al., 2012) and further, departing contributors may not provide notice and in an instant, may no longer be available for the purpose of knowledge transfer activities.

OSS projects have a hierarchical onion-like structure, consisting of core, co-developer, active users, and passive users (Crowston et al., 2004; Crowston and Howison, 2005; Mockus et al., 2002). The knowledge distribution in OSS projects is not uniform (Rigby et al., 2016). A small subset of contributors, typically 20%, called core members, make major code contributions of about 80% in OSS projects (Mockus et al., 2002). Absence of a contributor who is the original owner of the files or system on the project to perform maintenance tasks results in risking files to abandonment (Rigby et al., 2016). As depicted in

Figure 3.1 of the onion model, knowledge distribution in OSS projects is non-uniform with a higher concentration of code contributors in the centre of the onion than in the outer layers. The focus of this research is on the uniform distribution of knowledge among contributors, by the introduction of continuous knowledge transfer practices, which this work refers to as a proactive knowledge retention practices in OSS projects.

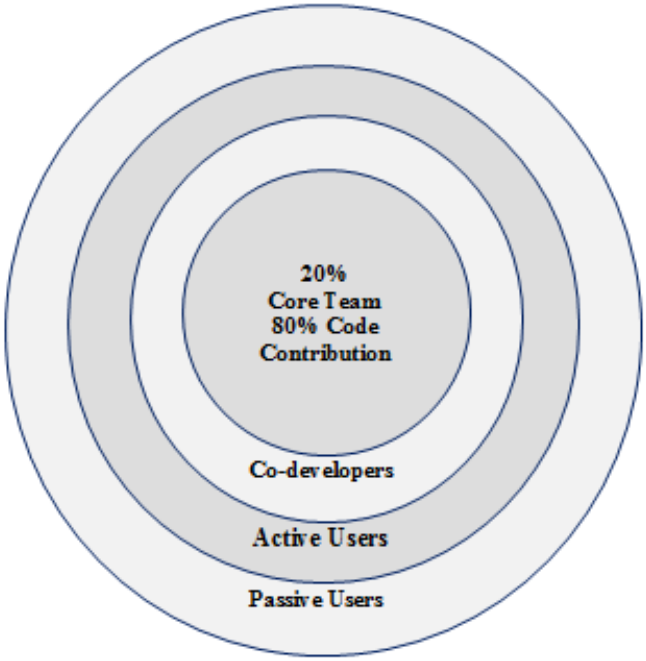

Figure 3.1: Onion model representing contributors in OSS projects users (Crowston et al., 2004; Crowston and Howison, 2005; Mockus et al., 2002)

3.3 Philosophical Background

Worldviews provide a general philosophical direction to research with common elements having different stances (Creswell and Clark 2011). Philosophical worldviews shape the approach taken for research by influencing research designs and research methods (Creswell 2014). Worldviews differ in: ontology which is the nature of reality; epistemology which refers to how we gain knowledge about what we know; axiology which explains role of values in research; methodology determines the process of research; and rhetoric is the language of research (Lincoln et al. 2011). The three worldviews reflected as

different philosophical concerns in research are positivism (also called post positivism), interpretivism, and pragmatism. Underlying philosophical concerns further determine the selection of the research method to conduct any research.

3.3.1 Positivism

The positivists advocate in the quantification of their learning through numbers and the use of statistical equations to predict human behaviour (Rubin and Rubin, 2011). A positivist believes that social life is pretty stable and constant (Denzin, 1973). In such an approach, if the learning of a concept is not possible through quantifiable methods it is generally ignored. The positivists extract simple relationships from a complex real world in numbers without considering the context (Denzin, 1973). Positivists stress on deterministic philosophy, reductionism, observation and measurement, and theory verification (Creswell, 2014). In deterministic philosophy, causes determine the effects or outcomes. Reductionism is about reducing ideas into small discrete tests consisting of variables based on hypothesis and research questions. Knowledge development by positivist is through observing and measuring objective reality in numbers, and by verification of laws and theories that govern the world.

3.3.2 Interpretivism

Interpretivist focuses more on human thoughts and actions in social and organisational contexts (Klein and Myers, 1999). Interpretivists (also called constructivists) believe in understanding the context and meaning by taking into account the real setting of the world in which they live and work. Interpretivist led research tends to develop subjective, varied, and multiple meanings about an object enabling them to unfold complex views, which are collected from many participants on the situation being studied (Creswell, 2014). Interpretivist in an open-ended questioning, a researcher carefully hears

all views of participants and shapes their interpretation from cultural and historical experiences (Creswell, 2014). Moreover, the outcome of a research led by an interpretivist generates or inductively develops a theory or pattern of meaning (Creswell, 2014).

3.3.3 Pragmatism

The third philosophical perspective, pragmatism, advocates an alternative world view to positivism and interpretivism and primarily focuses on the problem to be researched and the consequences of the research (Creswell and Clark, 2011). Pragmatism offers a middle position or mixed methods research movement with a practical and outcome-oriented method of inquiry based on action and leads by enabling researchers to have better answers to their research questions (Johnson and Onwuegbuzie, 2004). Pragmatism is about adopting a research approach that strikes a balance between positivism and interpretivism. Pragmatism takes a value-oriented approach to research and reach an agreement about importance of culturally derived values and desired conclusion (Johnson and Onwuegbuzie, 2004). Pragmatism as a worldview arises out of actions, situations, and consequences rather than preceding conditions. Instead of focusing on methods, researchers place a greater emphasis on the research problem and use all approaches available to understand the problem (Rossman and Wilson, 1985). Pragmatism as a philosophical underpinning for mixed methods studies, focuses attention on the research problem and uses pluralistic approaches to derive knowledge about the problem (Tashakkori and Teddlie, 2010) and is concerned with real-world practice. The three philosophical concerns with their main characteristics are summarised in Table 3.1.

Table 3.1: Three philosophical concerns

Positivism	Interpretivism	Pragmatism
- Determination - Reductionism - Empirical observation and measurement - Theory verification	- Understanding - Multiple participants meanings - Social and historical construction - Theory generation	- Consequences of actions - Problem-centered - Pluralistic - Real-world practice oriented

3.3.4 Research Philosophy Adopted

The philosophical position adopted in this research is that of pragmatism, with a focus on the research problem and enabling empirical research to find answers to the research questions presented in chapter 1. The pragmatist worldview reflects the direct action oriented approach of a researcher towards the investigation of the research problem at hand. This research would benefit by adopting pragmatism and approaching the problem by understanding it in a practical manner and by using multiple methods in research. As explained earlier in chapter 1, the goal of this research is to reduce KL by identifying KR best practices through systematic study and by engagement with practitioners. The identification of an overarching set of knowledge retention best practices requires understanding of the phenomenon and exploration in real life with multiple contexts and with the ability to quantify the concepts. The view taken in this research is that of taking a middle position between two extremities of being a positivist or an interpretivist. Solely being an interpretivist or positivist is inadequate in terms of addressing the objective of this research. The insights provided by the use of qualitative and quantitative research in a mix method can be integrated into a workable solution under pragmatism with an understanding that goal of mixed methods research is not to replace either qualitative or quantitative approach but to use their strengths and minimize the weaknesses of both in single research studies and across studies (Johnson and Onwuegbuzie, 2004).

3.4 Research Methodology and Methods

The terms research methodology and research methods are at times used interchangeably. Research Methodology is defined as "the collection of methods or rules by which a particular piece of research is undertaken" and as the "principles, theories and values that underpin a particular approach to research" (Somekh and Lewin, 2005). Another definition states: "methodology is the overall approach to research linked to the paradigm or theoretical framework" (Mackenzie and Knipe, 2006). In general, research methodology is a systematic approach to achieve particular goals of the research. Research designs under each research approach provide specific directions and guidelines to conduct research (Creswell 2014).

The research method refers to "systematic modes, procedures or tools used for data collection and analysis" (Mackenzie and Knipe, 2006). A method is mainly a set of principles through which empirical data is collected and analysed (Easterbrook et al., 2008). Research methods can be classified as qualitative, quantitative or both (Wohlin et al. 2003). The three types of research designs are quantitative (section 3.4.1), qualitative (section 3.4.2), and mixed methods (section 3.4.3).

3.4.1 Quantitative Research

"The quantitative research mainly focus on deduction, confirmation, theory or hypothesis testing, explanation, prediction, standardized data collection, and statistical analysis" (Johnson and Onwuegbuzie, 2004). Quantitative research is based on the use of statistical methods and establishes relationships between variables (Runeson and Höst, 2009) or quantifies a relationship by comparing two or more groups (Creswell, 2014). The aim is to identify a cause-effect relationship. Quantification is the confirmation of a hypotheses rather than the formation of the hypothesis (Sharif et al., 2015). Quantitative research employs

numbers to capture or describe some phenomenon but may be limited in the sense that it can miss (or overlook) certain important information (Rubin and Rubin, 2011). The strategies of inquiry employed in quantitative research are experimental designs such as true experiments and quasi-experiments, and non-experimental design such as surveys (Creswell, 2014).

- Experiments determine that if special treatment to a group can influence an outcome. Controlled experiments are quantitative in nature since they measure different variables and attain varying results. The variables are repetitively changed and measured again (Wohlin et al., 2003). The variables to be measured in quantitative research are identified based on the theory (Easterbrook et al., 2008). In controlled experiments a cause and effect relationship is studied by manipulating independent variables and observing the effect on dependent variables (Easterbrook et al., 2008). Controlled experiments are sometimes referred to as research in small (Kitchenham et al., 1995) and have a limited scope. The controlled experiments have a fixed design and the procedure to run an experiment is a formal one. Experiments are not a suitable research method for this work. The goal of this research as described in chapter 1 is to find best KR practices in OSS projects and to evaluate them for their effectiveness. The study of cause and effect relationship or intervention caused by changing variables to determine the influence on the outcome, does not resonate with the goal of this research.

- The primary means of gathering data are interviews or questionnaires which are sent to a large number of representing population (Wohlin et al., 2003). When conducting a survey, a sample of a representative population is selected based on criteria and results may be generalised for the sample population. The survey has to be designed carefully ensuring that the questions are understandable by the participant and the data

collected from the target population is valid and useful in servicing the hypothesis exploration and research questions. In this research questionnaire does meet the requirement of collecting data from a large number of practitioners in OSS projects to determine the effectiveness of KR practices.

Quantitative research focus on data collection that is largely of a numeric type and the required information is specified in advance and data is gathered using scaled instruments while interpretations are made on the basis of the statistical results (Creswell, 2014). The use of a questionnaire most likely includes a numerical rating scale for quantitative data collection (Johnson and Onwuegbuzie, 2004). In order to collect data, researchers can employ an instrument or test, which has a set of questions to evaluate the confidence towards an approach, or use checklist to identify and observe people involved in some task (Creswell, 2014). The specific questions asked from the participants are predetermined based on a set of variables. In this research a survey instrument detailed in chapter 5 has been employed to collect the qualitative data.

3.4.2 Qualitative Research

Qualitative research refers to the study of objects in their natural setting (Wohlin et al., 2003). A qualitative researcher tries to understand the causes while interpreting a phenomenon by accepting that there are multiple interpretations of the explanations given to them by the subjects in the study (Denzin and Lincoln, 2011). In qualitative research the subject is the person who participates in the study to evaluate an object and tends to interpret and understand a range of different views of the subjects on the concerned problem at hand (Wohlin et al., 2003). Case studies, ethnographies, post-mortem analysis, action research (Shull et al., 2008), phenomenology, grounded theory, and narrative (Creswell, 2014) are primarily qualitative in nature.

- A case study investigates a contemporary phenomenon in a real life context and the control is lower than with experiments. Case studies provides the richer and deeper description of the studied phenomenon (Runeson and Höst, 2009). A case study is an observational study on an ongoing project, while experiment is a controlled study (Easterbrook et al., 2008). It has been noted that case studies can combine data collection by using methods such as interviews, questionnaires, archives, and observations (Eisenhardt, 1989). Similarly, a case study can also embed other research methods e.g. a survey may be conducted within a case study (Runeson and Höst, 2009). The use of mixed method approach explains that survey can be used within a case study (Yin, 2013). In this research, case study can be a candidate research method for investigation. The case study requires selection of some OSS projects in order to find KR practices and send questionnaires to OSS contributors. The goal of the research is to design KR practices and attain feedback from OSS contributors on the effectiveness of KR practices and it should not be limited by the selection of certain OSS projects than others.

- The post-mortem analysis method helps to learn from past experience and improve software development process by consulting the projects' documents and interviewing people who were involved with the object under analysis, for example, a project (Wohlin et al., 2003). A post-mortem is normally conducted close to finish an activity or project and analyses can be performed for projects that are still underway using this retrospective approach (Wohlin et al., 2003). Post-mortem research limits the selection of OSS projects and is not considered as a research approach in this work.

- Ethnography is another research method applied in a participant-observer manner (Shull et al., 2008). The researcher participates with the team members to observe and understand the social interaction taking place in

the community (Easterbrook et al., 2008). Employing ethnography again limits the researcher to interact with only limited number of OSS contributors to design KR practices. The results on the effectiveness of KR practices will only be from a small number of OSS contributors, which are not sufficient to meet the goal of this research.

- In action research, the researcher applies an iterative problem solving approach to a current situation to improve it (Easterbrook et al., 2008). The action research method requires an agreement from the problem owner(s) to collaborate and identify the problem. Further, the problem owner(s) should be involved to solve the problem (Easterbrook et al., 2008). Action research is not considered to be a suitable research approach for this work considering the limitations imposed on the researcher to find specific OSS project for collaboration. Every OSS project is different in work setting, domain, and varies in rules and policies implemented. It is not feasible for the researcher to find enough projects to design KR practices or find adequate number of OSS contributors to evaluate KR practices.

- Narrative research is a design from humanities where the researcher asks one or more than one individual to share stories of their lives (Riessman, 2008). This information is then retold in the form of a narrative in chronological order (Creswell, 2014). The narrative combines life view of an individual shared as stories with those of the researcher's life as a collaborative narrative (Costantino, 2001). Narrative research does not fulfil the goals of this research to design KR practices or to evaluate their effectiveness in OSS projects.

- Phenomenological research is a design originating from philosophy and psychology where the researcher describes the experiences of participants about a phenomenon as provide by participants (Creswell, 2014). The

details provided on experiences of several participants about a phenomenon develops into a core description of a phenomenon under study (Creswell, 2014). Phenomenological design strongly underpins the philosophical attributes of a research topic and involves conducting interviews (Giorgi, 2009). This work does relate to phenomenological research aspects by inquiring OSS contributors of their experience through survey interviews in the form of a questionnaires.

- Grounded theory is another qualitative research design of inquiry from sociology, which facilitates the researcher in deriving a general, abstract theory about a process, action, or interaction grounded in the views of participants (Creswell, 2014). The data collection involves using multiple stages until a data saturation point is reached and no new information is found, the data refinement is through interrelationship of categories of information (Strauss and Corbin, 1998). The principles of Grounded Theory are employed, during the analysis of the qualitative data using coding, memoing and constant comparison (chapter 4, section 4.5 and 4.6) to develop PKR canonical model. The development process of the PKR canonical model is detailed in chapter c4 section 4.2.

Surveys, case studies and post-mortem analysis can be classified as both quantitative and qualitative based on the design of the investigation (Wohlin et al., 2003). The survey can be qualitative or quantitative based on the questionnaires design, type of data and application of any statistical methods (Wohlin et al., 2003). Survey conduction can be in retrospect or can be launched before the execution of a project where the objective is to get some ideas of the outcome of the forthcoming project (Wohlin et al., 2003).

Data collection in qualitative research involve observing the behaviour of individuals and conducting interviews with individuals where they can talk about a topic openly mostly without the use of specific questions. The

information in qualitative data collection emerge from the participants of the study with the data in textual form or recordings (Creswell, 2014). Researchers make interpretations from the themes or patterns that emerge from the data. Qualitative research theory may be applied after the data collection while following the process of coding (data analyses by labelling and categorising) (Easterbrook et al., 2008).

3.4.3 Mixed Methods Research

"Mixed methods research is formally defined as the class of research where the researcher mixes or combines quantitative and qualitative research techniques, methods, approaches, concepts or language into a single study" (Johnson and Onwuegbuzie, 2004). Mixed methods involves combining qualitative and quantitative research by the collecting qualitative data, which is open-ended and without predetermined responses, and quantitative data, which is closed-ended in nature and the selection of responses is from predetermined list of answers (Creswell, 2014). The mixed methods are valued and thought to neutralize the weakness and bias that arises in research by the usage of just one method (Creswell, 2014).

Mixed methods design can vary based on the combination of quantitative and qualitative research as depicted in Figure 3.2. Design 1 and 8 are mono method designs with the use of only one research method decided with research objectives and carried throughout the data collection and analysis stage. The mixed-model designs are depicted by design 2, 3, 4, 5, 6, and 7.

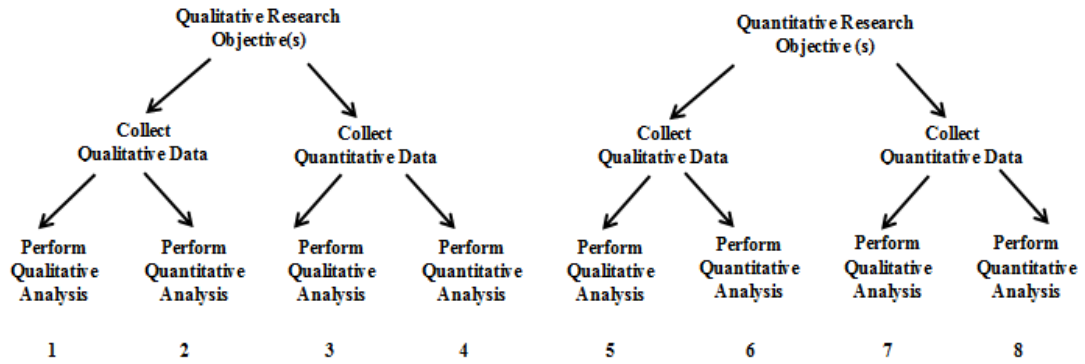

Figure 3.2: Monomethod (1 and 8) and mixed-model designs (2 to 7) (Johnson et al., 2007)

The mixed method data collection comprises of both types of data quantitative and qualitative, to measure the concepts, parallel variables, or constructs, under study and have the following characteristics (Creswell, 2014):

- Both kinds of data is collected such as qualitative, which is open-ended and quantitative, which is closed-ended data to investigate research questions or hypotheses.
- The qualitative and quantitative data analysis is conducted for both types of data. The procedures for both qualitative and quantitative data analysis are detailed to perform analysis with adequate sampling, sources of information, data analysis steps.
- The qualitative and quantitative data are integrated during the analysis through merging the data, connecting the data, or embedding the data.
- The mixed method also incorporates the timing of both types of data collection such as concurrent or sequential and the proportion of each type of data collection such as equal or unequal.

3.4.4 Research Design Adopted - Mixed Methods Research

The pragmatist worldview adopted in this research lays the foundation to the research methodology including the research design. Pragmatically inclined researchers focus more on the desired outcomes and solution to the problem. Mixed methods research applies pragmatist system of philosophy where "the researcher mixes or combines quantitative and qualitative research techniques, methods, approaches, concepts or language into a single study" (Johnson and Onwuegbuzie, 2004).

In this work, mixed method research is employed to investigate the research questions stated in section 1.1 and reiterated in the begining of this chapter. The objective of this research is to systematically investigate knowledge loss and mitigate its implications in OSS projects. The goal is to reduce KL by identifying knowledge retention best practices through systematic study and by engagement with practitioners. Mixed method research emerges from pragmatism, a worldview that offers to combine the positivist view of quantifying the object of research and the interpretivist view of interpreting different meaning associated with it. Therefore, both research types, qualitative and quantitative, which merge in mixed method research design, serve the purpose of understanding the phenomenon of knowledge retention in OSS projects and generalizing the results to a larger community of OSS projects. This research adopts a mixed method approach with qualitative and quantitative data collection and an independent data analysis for each data type. The findings from analysis will be merged to understand and elaborate the phenomenon of proactive knowledge retention in OSS projects. There are five major rationales related to the decision of researchers to conduct mixed method research (Johnson et al., 2007):

- Triangulation is validation of results obtained from different methods and designs while studying the same phenomenon

- Complementarity is the explanation, enhancement, illustration, and clarification of the results obtained from one method with the results from the other method
- Initiation is the reframing of the research question by discovering inconsistencies and contradictions
- Development of method by using the findings from one method to update the other method
- Expansion is extending the range of research by using different methods for different inquiry components

In this research, the rationale of using mixed method research is related to triangulation, complementarity, and expansion. Using qualitative and quantitative methods will facilitate a triangulation of results from different perspectives on the phenomenon of knowledge retention in OSS projects. Complementarity will further play an essential role to use one method to elaborate and clarify the results obtained from the other method i.e. complementing the results of quantitative methods with the help of the qualitative method. Using different methods for different inquiry components will also expand the breadth and range for descriptive research. At a practical level, mixed methods has its strength, for utilising both qualitative and quantitative research to overcome the limitation of each approach; at a practical level the mixed method approach is complex and sophisticated; and at a procedural level, a mixed method approach provides offers the potential to establish a more rounded understanding of the problem (Creswell, 2014).

The data collection for this research adopts mixed methods that involve both data types, i.e. qualitative data and quantitative data. The data collection for the purpose of developing PKR canonical model involves qualitative data. Further, to evaluate the PKR canonical model a survey instrument is employed for qualitative and quantitative data collection. The purpose of data collection

using the survey is to evaluate the proposed set of practices in proactive knowledge retention canonical model for their effectiveness and improvement based on the feedback from the OSS community. The PKR canonical model itself is developed following a rigorous process, the development process is entailed in chapter 4 section 4.2. The data collection to develop PKR canonical model is through the selection of data components entailed in section 4.3. The PKR canonical model forms the basis to develop survey instrument to be employed for data collection from contributors in OSS projects as discussed in chapter 5.

The quantitative and qualitative data collection through survey instrument is performed in parallel. Rather than using one type of data collection the combination of both quantitative data (close-ended questions) and qualitative data (open-ended questions) overcomes the weakness arising from the use of only one data type.

3.4.5 Data Analysis

The textual data collected for the development of PKR canonical model is analysed based on the Grounded Theory techniques of coding, memoing, and constant comparison further explained in chapter 4, section 4.5. For the qualitative and quantitative data collected through survey, there are different possible ways to analyse the data. The data collections can be merged by a side-by-side comparison, by transformation procedure or by table or graph (Creswell, 2014). In side-by-side comparison, the results from both types of data collection are reported separately and then findings are compared. In transformation procedure, the qualitative data is transformed to quantitative data by changing the qualitative themes into quantitative variables. The two types of data collection can also be merged in the form of tables and graph while displaying both forms of data and merging them in a single visual. This research adopts side-by-side comparison and analyse the qualitative and quantitative data collections separately and then results will be combined to

incorporate the feedback from OSS contributors into PKR canonical model.

Table 3.2 summarises the three types of research methodologies and specific details on research design, data collection, and analysis.

Table 3.2: Summarising three types of research methodologies

Research Methodology Includes	**Qualitative Research**	**Quantitative Research**	**Mixed Method Research**
Philosophical Worldviews	Interpretivism (Wohlin et al. 2003; Denzin, 1973)	Positivism (Rubin and Rubin 2011; Creswell, 2014)	Pragmatic (Creswell and Clark 2011; Tashakkori and Teddlie 2010)
Research Designs or Method of Inquiry	Case studies, Action Research, and Ethnography (Shull et al. 2008) Phenomenology, Grounded Theory, Ethnography, Case Study, and Narrative (Creswell, 2014)	Experiments and Quasi-Experiments, and Non-Experimental (e.g. Surveys) (Creswell, 2014)	Sequential, Concurrent, and Transformative (Creswell, 2014)
Surveys, Case Studies and Post-mortem Analysis can be both Quantitative and Qualitative (Wohlin et al. 2003)			
Research Methods or Data Collection	Open-ended Questions, (Creswell, 2014) Interview Data, Observation Data, Document Data, and Audio-Visual Data	Closed-ended questions, predetermined approaches, numeric data (Creswell, 2014) Performance Data, Attitude Data, Observational Data, and Census Data	Both open- an closed-ended questions, both emerging and pre-determined approaches (Creswell, 2014) Multiple Forms of Data
Analysis	Qualitative Analysis	Quantitative Analysis	Both Qualitative and Quantitative Data analysis (Creswell, 2014)

3.4.6 Validity Concerns in Mixed Method Research

There are key validity concerns while using the mixed method research that uses both qualitative and quantitative data. The first one is of unequal sample size when qualitative and quantitative data is collected (Creswell, 2014). Generally quantitative sample size is larger than qualitative sample size to perform statistical tests. The sample size for a qualitative data collection is small since the intention is to study the sample extensively and gain an in-depth perspective. The second concern is that it can be problematic to compare the findings from two data types with different variables and merge them will lead

to incorrect strategy for inquiry (Creswell, 2014). The third concern is to have the same participants in qualitative and quantitative data collection for a better comparison (Creswell, 2014).

In relation to the first concern, the qualitative and quantitative data collection is conducted simultaneously through survey questionnaire from each participant in such a way that both data types are represented equally in the sample size. For the second concern, the two types of data collection are analysed separately by using side-by-side comparison. The results of both analyses will be merged and reported collectively. To account for the third concern, the design of the survey questionnaire ensures that same concepts are measured in collection of both qualitative and quantitative data by same set of participants removing any inconsistencies.

3.5 Empirical Research

Empirical studies are emphasised to provide a scientific basis for software engineering Wohlin et al. (2003) and to investigate the social and cognitive processes surrounding complex software systems (Shull et al., 2008). Empirical methods allow for informed and well-grounded decisions and allow the investigation of a phenomenon by experimenting and experiencing it in the real world settings (Wohlin et al., 2003). Empirical research employ surveys, case studies and post-mortem analysis as strategies of inquiry under both qualitative and quantitative approach (Wohlin et al., 2003). Typically, case studies, action research, and ethnography are primarily qualitative in nature (Shull et al., 2008).

In this research, mixed method approach is adopted to empirically investigate KL due to contributor turnover by designing and evaluating PKR canonical model for its effectiveness in OSS projects. The development of PKR canonical model follows a rigorous process and is described in chapter 4. OSS

projects are diverse in nature and vary intrinsically in their governance, policies, organisational structure, domain, and software development practices. In order to investigate KR in OSS project, the data collection from only a few OSS projects will not serve the purpose of this research.

Similarly, the nature of examination requires that the findings be validated by collecting more than one type of data i.e. qualitative and quantitative. Mixed method research allows collection of both types of data in this research by employing surveys. The development of the survey instrument, the selection of the survey participants from OSS community and execution of the survey to collect data is discussed in chapter 5.

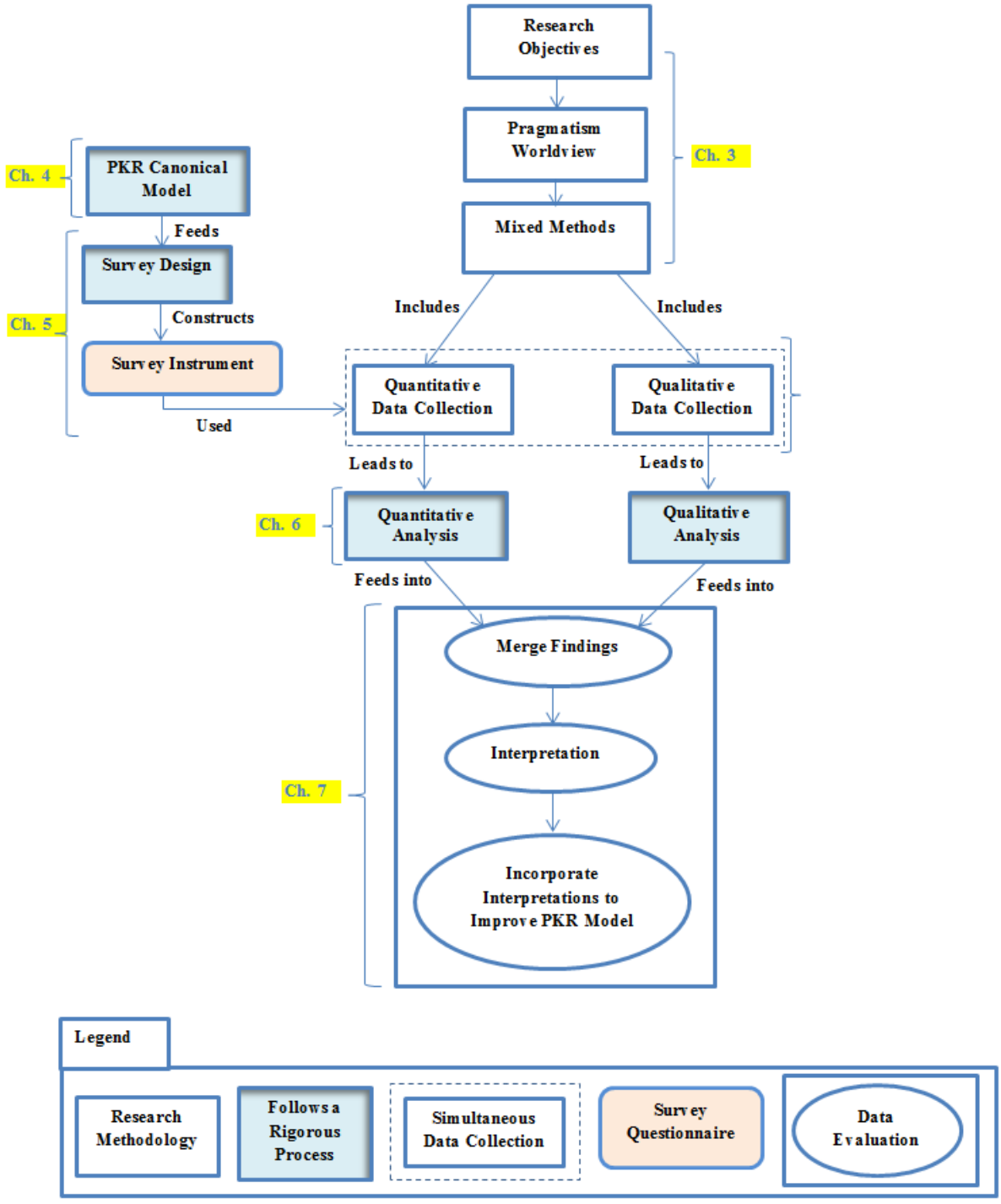

Figure 3.3: The overall research methodology and road map for chapters 4-7

The data collected through the survey to evaluate the effectiveness of PKR canonical model is analysed in chapter 6. The evaluation and research findings are presented in chapter 7. Figure 3.3 depicts the overall research methodology including and the road map for chapters 4 to chapter 7.

3.6 Chapter Summary

The worldview embraced in this research is of pragmatism, which closely relates with the research objectives of investigating proactive knowledge retention strategy in OSS projects. A pragmatist takes an action-oriented approach with an outcome of the research. A pragmatist takes a middle position between a positivist and an interpretivist. To examine the objectives of this research under a pragmatist worldview, mixed method approach will be applied. The mixed method research design will allow concurrent data collection of both qualitative and quantitative types to follow with analysis. In the case of the data collected through survey instrument the findings from the analysis of qualitative and quantitative data will be merged for a better understanding of the phenomenon.

The development of survey instrument and data collection is detailed in the chapter 5. The responses from the survey questionnaire will be used to evaluate and improve the proposed PKR practices in OSS projects. The PKR canonical model developed by employing qualitative research, presents a set of proposed KR practices, which underpins the design of the survey instrument presented in chapter 5. The next chapter discusses the development process of PKR canonical model.

Chapter 4

Proactive Knowledge Retention Canonical Model

4.1 Introduction

The purpose of this chapter is to elaborate the development process of Proactive Knowledge Retention (PKR) canonical model for OSS projects. The various definitions relating to the term "canonical" are: “established or well-known or widely recognised as a model of authority or excellence” [1]; “conforming to a general rule or acceptable procedure” [2]; “authoritative or standard; conforming to an accepted rule or procedure” [3]; and “the term canonical depicts the standard state or manner of something” [4]. In this work, PKR canonical model represents a set of practices, which conform to the dynamic work structure in OSS projects and can be effectively used as a well-known or widely recognised model of excellence. The PKR canonical model evolved based on the belief of being proactive in knowledge sharing and transfer among contributors, while they are still active in OSS projects. The earlier literature review on knowledge loss due to contributor turnover in OSS projects led to the identification of knowledge retention mitigation techniques detailed in chapter 2, section 2.12 that can prove beneficial to alleviate knowledge loss in OSS projects.

[1]https://www.vocabulary.com/dictionary/canonical
[2]https://www.merriam-webster.com/dictionary/canonical
[3]https://www.webopedia.com/TERM/C/canonical.html
[4]https://www.techopedia.com/definition/1320/canonical

A consolidated view of PKR practices based on literature in OSS and in organisations is the baseline for the formulation of the PKR canonical model. The PKR canonical model intends to transform contributor's use of knowledge and engagement in knowledge relevant activities including knowledge sharing and knowledge transfer.

In the sections to follow, the details on the canonical model development process are presented, which include selection of data components, data preparation, and analysis using principles of grounded theory and finally an elaboration on the developed PKR canonical model.

4.2 Canonical Model Development Process

In order to develop the PKR canonical model, the candidate details are selected as data components from the data sources, which are further refined and classified into categories by applying the principles of grounded theory. Data in the context of PKR canonical model refer to all textual descriptions selected for the construction of PKR canonical model. A data component refers to one textual description selected from one of the three data sources and is considered as one unit among all data. Data contains all data components employed for the construction of PKR canonical model.

The canonical model was developed in three phases, selection of data components, preparation of data, and data analysis. In the first phase, data components were selected from three data sources as the starting point for the development of the canonical model elaborated in section 4.3. In the second phase, the data collected from three sources is prepared and refined by following a sequence of steps explained in section 4.4. In the third phase, the collected data is analysed through the principles of grounded theory as explained in section 4.5 and 4.6. The complete process followed to develop the PKR canonical model is illustrated in Figure 4.1.

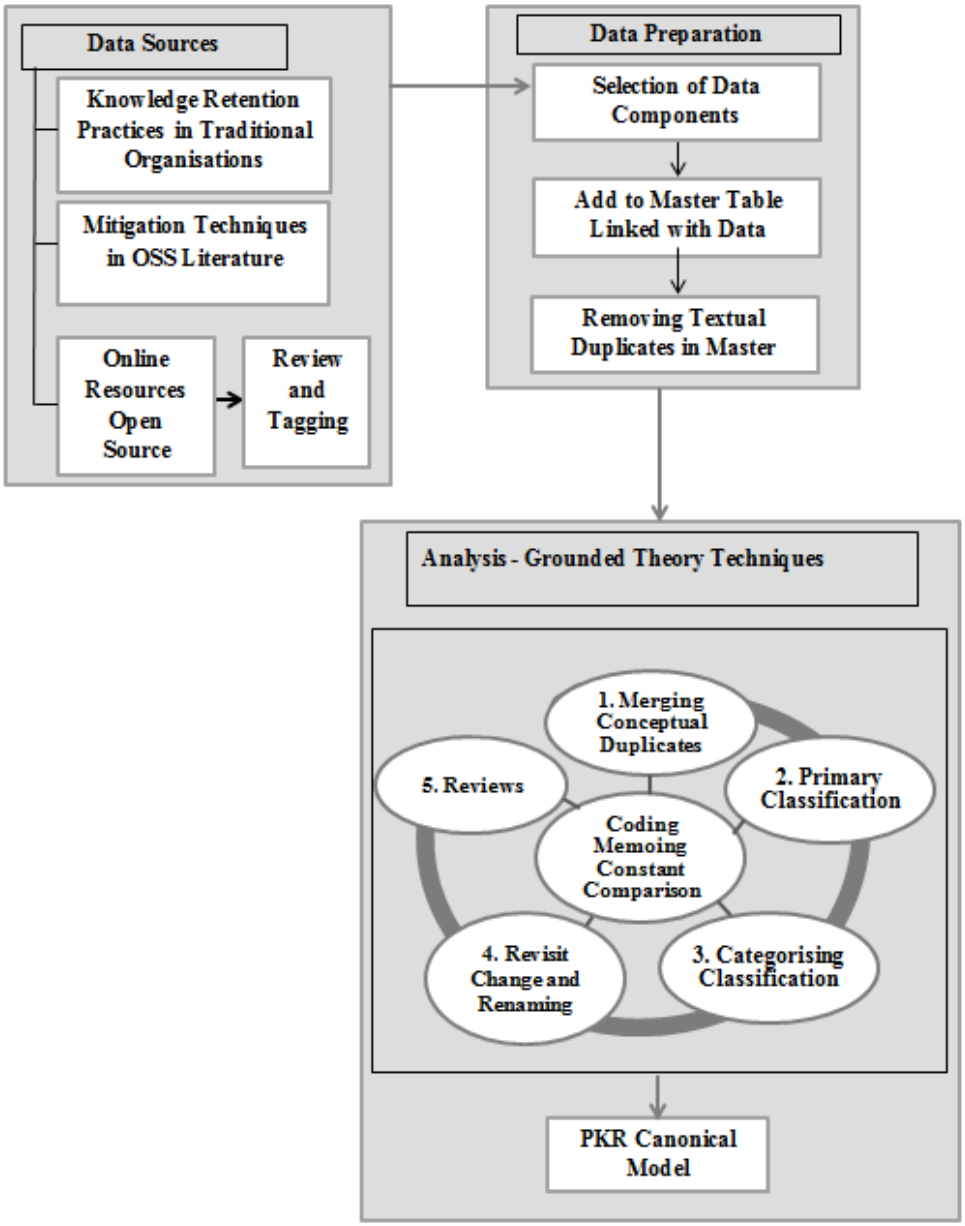

Figure 4.1: Development process of PKR canonical model in OSS projects

4.3 Selection of Data Components

This section elaborates the selection of data component fundamental to the construction of PKR canonical model. In order to develop the canonical model the selection of data components is from three data sources: (1) KR practices employed in organisations detailed in sections 4.3.1 and 4.3.2, (2) knowledge loss mitigation techniques presented in OSS literature, and (3) open source

guides online. The requirement of the first data component is fulfilled by the identification of KR literature on organisations. Identification of knowledge loss mitigation techniques is from OSS literature entailed in chapter 2. Additionally, "OSS guides" were identified from online resources, which aggregates best practices from OSS community. Moreover, any existing online resources were also consulted to find any additional practices relevant to OSS projects[5].

4.3.1 Knowledge Retention in Organisations

The competitive advantage of organisations currently lies particularly in the application of employee knowledge. The continuity of an organisation's functioning is dependent on the experience and skills. Knowledge continuity ensures elimination of the KL threat during employee turnover and other personnel changes (Urbancová and Linhartová, 2011). KR can be seen as a way of embedding and enabling knowledge within an organisation and a critical factor for sustainable performance (Doan et al., 2011). KR strategies used in an organisation advance innovation, organisational growth, efficiency, employee development, and competitive advantage. The three categories of KR initiatives in organisations identified in the literature are (Leibowitz, 2011):

1. Recognition and reward structure.
2. Knowledge personalization and codification.
3. Retaining key employees.

Recognition and Reward Structure - In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the organisation. Furthermore, to encourage contribution of knowledge, based on codification and personalisation a reward system is established for people documenting and sharing knowledge

[5]https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

(Hansen et al., 1999). A reward structure is based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition. Extrinsic motivation is related to monetary incentives (Gammelgaard, 2007). Organisations like Xerox, and Hewlett-Packard reward people for sharing their knowledge (Rus et al., 2002). A reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer, 2003). Using the combination of extrinsic and intrinsic rewards is (Gammelgaard, 2007). In the two types of reward structures, the long lasting one is intrinsic reward structure. As an example of intrinsic motivation, Google consists of a user community mainly of software engineers, where the knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated, thus software engineers willingly share their knowledge. Even though the technology changes very quickly, capturing the gained knowledge still is worth the effort (Rus et al., 2002).

Personalisation and Codification - Codification and personalisation are also considered as useful strategies for knowledge bases to be further used in knowledge intensive activities like software development (Donnellan et al., 2005). Codification is the documentation of the knowledge to be stored, disseminated, and reused. Personalisation is the development of network to connect people where they can share tacit knowledge (Hansen et al., 1999). In knowledge bases, codification captures electronic information and personalisation deals with the ways humans use and process knowledge (Donnellan et al., 2005). Organizations implement codification strategy to encourage the reuse of explicit knowledge. Figure 4.2 illustrates two strategies namely personalisation and codification employed by organisations to manage knowledge (Jansen, 2008). In a personalisation, the nature of the most

knowledge in the organisation remains tacit. The only knowledge that is made explicit is about people who have the information. The left pyramid in the figure depicts personalisation strategy, the knowledge pyramid is broad for the tacit knowledge and becomes smaller for the documented and formal knowledge. On the contrary, an organization following a codification strategy, codifies and stores explicit knowledge in documents and databases. Personalisation relates to connecting people, which includes tools such as mentoring, job rotation, knowledge fairs and communities. While codification involves tools used for after action reviews such as various knowledge repositories, and lessons learned system (Liebowitz, 2008).

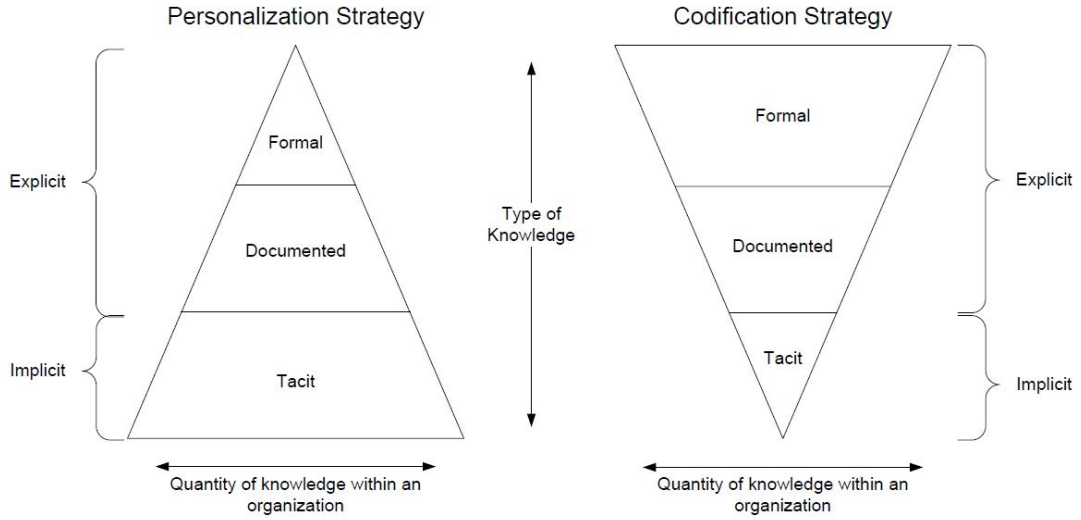

Figure 4.2: Pyramid of knowledge types and associated Knowledge Management (KM) strategies (Jansen, 2008)

Retaining Key Employees - The key employee can be retained by implementing phased retirement systems. The key employees after the retirement can be retained through programs such as leave of absence, part-time work, and temporary jobs. Bringing back retirees in different capacities can also help to retain knowledge (Leibowitz, 2011).

4.3.2 Knowledge Retention Practices in Organisations

The knowledge retention techniques in organisations were briefly discussed in reference to knowledge conversion in chapter 2, secion 2.10. As one of the data source to construct canonical model, this section delves into more detail on KR practices in organisations as presented in the literature.

The core techniques designed to retain knowledge in a organisation are mainly dependent on its knowledge sharing practices. There are many techniques that facilitate knowledge capture, sharing, and reapplication namely after-action reviews, communities of practice, face-to-face meetings, mentoring programs, expert referral services, video conferencing, interviews, written reports, use of training or and technology-based systems to transfer the knowledge (De Long and Davenport, 2003). Knowledge sharing should be encouraged in organisations by documenting and storing knowledge in a KM repository whenever contributors collaborate to resolve issues. The knowledge, when recorded, can help resolve issues for others facing similar problem (Rus et al., 2002). Tacit knowledge is transferrable with practices involving more face-to-face interaction, such as mentoring and after-action reviews (post-mortem). Whereas in explicit and rule based knowledge, the less face-to-face interaction is required, making it more feasible to use training or technology-based systems to transfer the knowledge.

Technology is often used to enable the set of supporting KM activities in software engineering such as supporting document, competence management, and software reuse rework (Rus et al., 2002). Relevant technical tools allow for the retention of supporting documents, and competence management and used by software organisations for supporting software engineering practices along with KM. For example, document management tools frequently employed in organisations are Hyperwave, Microsoft SharePoint, Lotus Domino, and Xerox DocuShare. Similarly, some organisation use expert management systems to

identify experts. Competence management systems such as SkillScape, SkillView allow expert to add their profiles. KnowledgeMail is another tool that automatically generates competence profiles by assuming that people's e-mail and document reflect their expertise. Further, the activity of software reuse enables programmers to reuse the existing software in repositories to reduces the rework (Rus et al., 2002).

A) Interviews and Trainings

Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing (De Long and Davenport, 2003). Interviews are considered as an expensive approach to knowledge capture. Delta Airline and Tennessee Valley Authority interview their employees retaining critical specialised knowledge. NASA has commissioned some senior-level retirees to write case studies.

B) Storytelling

Story telling can play a valuable role in transferring knowledge to convert tacit knowledge into explicit knowledge (De Long and Davenport, 2003). It is not found useful for fact based technical tasks. NASA, the World Bank, and IBM made productive use of storytelling, and there is ongoing research in this area.

C) Community of Practices

Community of practices are used as a long term strategy by management to retain knowledge before it is lost (De Long and Davenport, 2003). Companies like Buckman and Shell have experience in building distributed global communities, which is found challenging in geographically distributed organisations. Siemens and BMW to resolve the challenges of losing expertise through retirement and attrition found a cross-company community network. Intel, Infineon Technologies, and Winterthur Insurance Switzerland joined the "Leaving Experts Community of Practice". Members of the community must be

experienced in dealing with the problem of losing expertise and be able to contribute valuable lessons to member companies. Furthermore, they cannot be a competitor to any member company.

D) Experience Based Memory

Rus et al. claim that software engineers in their daily work environment based on their experience create project and product memory (Rus et al., 2002). Version control, change management, documenting design decisions, and requirements traceability are software engineering practices that help build project and product memories as an indirect or direct effect of software development. One way to manage knowledge built on experience suggested is to divide the responsibility of capturing and reusing experience, between separate parts of development organisation as in "Experience Factory" (EF). EF builds on the technical and social KM infrastructure and reuses life cycle experience, processes, and products. Experience is collected from software development projects and are packaged and stored in an experience base (Dingsøyr and Conradi, 2002). The EF analyses includes resources such as all experience types, lesson learned, project data, and technology reports. EF also provides repository service for these resources. The EF supports software development by providing tailored experiences including valuable knowledge. Ericsson implemented a version of EF called experience engine, which relies on tacit knowledge rather than experience stored in experiences bases (Rus et al., 2002).

E) Job Rotation

In software organisations, job rotation enhances knowledge exchange and transfer where people work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains (Fægri et al., 2010). It is argued that the approach of job rotation adds unnecessary redundancy and incurs cost unjustifiable in an organisation (Fægri et al., 2010). Job rotation has positive effect of job on knowledge acquisition (Santos et al. 2016). Some development practices, such

as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation (Rus et al., 2002).

F) Mentoring

Mentoring is suggested to be a logical approach for transferring important tacit and implicit knowledge. It is argued that most companies due to resource limitations and an inadequate availability of experts find mentoring difficult to sustain in long term (De Long and Davenport, 2003). For example, Quest has adopted a "buddy approach" where experts are involved to mentor younger employees.

G) Post-mortem

Post-mortem reviews are a practical method to capture and reuse experience from projects that are complete or have finished a major activity phase (Dingsøyr, 2005). The intention of a post mortem review is to learn through a collective activity and to build knowledge based on the experience to improve future practice. The tangible outcome of a post-mortem is a report. In Apple a similar approach is used where results are published based on a project survey, collecting useful information on project, a debriefing meeting, and a 'project history day'. Microsoft invests considerable efforts into writing 'Post-mortem reports'. The reports contain details on what has worked well for the last project, what has not worked well and teams that need improvement for the next project. Dingsøyr relate post-mortem to Nonaka and Takeuchi model of combination of learning through socialisation (Dingsøyr 2005). In post-mortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Post-mortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects (Huber, 1996).

This section discussed that organisations worldwide utilise KR practices to convert tacit knowledge to explicit knowledge. The technology is used to store experience-based knowledge of the employees in the knowledge systems as the outcome of the process used to convert tacit knowledge to explicit. Experiences and knowledge of employees are valued, and resources are invested in organisations to capture it effectively. KL due to contributor turnover is globally recognised in organisations operating in diverse domains. KR is enabled in organisation using variety of approaches such as post-mortem, experience-based memory, interviews, trainings, job rotation, storytelling, mentoring, and learning based on communities of experts. The mentioned KR practices used in organisations to reduce KL are selected data components contributing towards the development of PKR canonical model.

4.3.3 Knowledge Retention Mitigation Techniques and OSS Guides Online Resources

In chapter 2, section 2.12, KR mitigation techniques in OSS projects were discussed. These techniques are also one of the data sources for data components utilised for the construction of the PKR Canonical Model.

Open Source Guides were created and are curated by GitHub along with input from outside community reviewers, but they are not exclusive to GitHub products (Eghbal et al., 2018). The goal of open source guide to aggregate community best practices, not what GitHub (or any other individual or entity) thinks is best with the intention to accumulate enough resources for people creating open source projects (Eghbal et al., 2018). Open source guides focus on six aspects in open source community including how to contribute to open source, starting an open source project, finding users for your project, building welcoming communities, best practices for maintainers, and leadership and governance. The review and tagging technique is thoroughly applied to the OSS guides online. Consequently,

key KR practices suitable for OSS projects were highlighted.

4.4 Data Preparation

The three data sources are explained in section 4.3. In order to construct a PKR canonical model a systematic approach is required to consolidate data components from three data sources by traversing and capturing the real essence of the intertwined articulated concepts. In this section, details follow on the selection of data components from three data sources. Data used in the construction of PKR canonical model is of textual nature. The data component refers to the textual description selected from three data sources mentioned in section 4.3.

Table 4.1: Master table with data components and link to sources

Component No.	Data Components	OSS Literature	Traditional Organisations	Online Open Source Guides + Others
1	Visualisation of Resources- A visualization word cloud has been proposed to show quickly the level of cooperation of the team in the project [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause	x		
2	A successor is a person who has relevant expertise and is knowledgeable on the work of other contributors. Identification of successors and involving them as co-owners is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A	x		
3	Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95].	x		
4	Centralisation: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers	x		

Data components are extracted while keeping a track to original sources in the master table as illustrated in Table 4.1. Consolidation of all data components in the master table while keeping track tot the source is the preliminary step

before the application of a specific approach for data analysis resulting in the PKR canonical model. KR practices in organisations are discussed in section 4.3.2. The details on each KR practice is added as a data component to the master table with an 'x' indicating the source of the data component against the column name, e.g. 'literature from organisations'. Initially 11 different KR data components from literature on organisations are added to master table. The 13 data components are identified from KL mitigation techniques in an extensive OSS literature review in chapter 2, section 2.12 and mentioned in section 4.3.3 as one of the data source, were added to the master table. The selected set of 31 data components from open source guides were also added to the master table with an indication to the source, e.g. open source guides. In all, there are 55 data components from three data sources are consolidated in the master table.

In the next step, the master table containing data components is searched for the instances of textual duplication. Textual duplication are the details existing in the master table more than once. The details of data components are compared to identify instances of text with similar meaning and descriptions. The elimination of duplication activity, does not require any documentation and only requires appending an additional data source against the data component in the master table. There was one instance of textual duplication observed relevant to mentoring, which was common in two data sources, i.e. traditional organisations and OSS guides. The final outcome of removing textual duplication was remaining 54 data components in the master table. The complete master table containing 54 data components along with data sources appears in appendix B.1.

4.5 Analysis - Principles of Grounded Theory

The details of practices gathered in the master table require a systematic translation into a canonical model through Qualitative Data Analysis (QDA) .

QDA tends to be inductive in nature and the qualitative data analyst identifies important categories in the data, as well as patterns and relationships, through a process of discovery (Schutt 2011). The textual data is described in ways that capture the setting or people who produced this on their own terms rather than in terms of predefined measures and hypotheses (Schutt 2011). Grounded theory principles are utilised to conduct QDA and to develop the PKR canonical model with rigour as illustrated in Figure 4.1. Glaser's definition of grounded theory is, a general methodology of analysis linked with data collection that uses a systematically applied set of methods to generate an inductive theory about a substantive area (Glaser, 1992). Grounded theory emphasises the systematic approach to data collection, handling and analysis (Douglas 2003). The canonical model is systematically developed using coding, constant comparison, and memoing from the principles of grounded theory.

In Grounded Theory, data analysis involves what is commonly termed coding, "taking raw data and raising it to a conceptual level" (Corbin and Strauss, 2008). Coding involves interacting with data using techniques such as asking about the data, making comparisons between the data, and in doing so, deriving concepts to stand for those data, then developing those concepts in terms of their properties and dimensions (Corbin and Strauss, 2008). Open coding refers to the process of breaking down, conceptualising, and re-assembling data while interrogating data line by line, by sentence or by paragraph, or as a document in its entirety, using comparison (Corbin and Strauss, 2008; Holton, 2007). It is a form of content analysis that is used to find and conceptualise the underlying issues amongst the noise of the data (Allan, 2003). Coding is the central process by which grounded theories are derived and which enables the analyst to think with an open mind and overcome inherent biases during the research process. The codes obtained are grouped together based on a theme to form concepts and are not used as labels under which similar instances of the same phenomenon are counted. Moreover, "codes capture patterns and themes and cluster them under a 'title' that evokes

a constellation of impressions and analyses for the researcher" (Lempert, 2007).

Constant comparison refers to "the analytic process of comparing different pieces of data for similarities and differences" (Corbin and Strauss, 2008). This method of analysis "generates successively more abstract concepts and theories through inductive processes of comparing data with data, data with category, category with category, and category with concept" (Bryant and Charmaz, 2007). This type of comparison is considered essential to all analysis because it allows the researcher to differentiate one category or theme from another and to "identify properties and dimensions specific to that category or theme" (Corbin and Strauss, 2008). In the case of this research, the various data codes identified in the selected data components must be compared for similarities and differences through constant comparison until a final set of categories emerge. Here the constant comparison is applied in successive iterations producing a better version of practices under each category.

The coding of data in Grounded Theory occurs "in conjunction with analysis through a process of conceptual memoing, capturing the theorist's ideation of the emerging theory" (Bryant and Charmaz, 2007). Memos can be considered to be "written records of analysis" and during the process of memoing, a certain degree of analysis occurs and the very act of memoing "forces the analyst to think about the data and it is in thinking that analysis actually occurs" (Corbin and Strauss, 2008). Memos are both a methodological practice and a simultaneous exploration of processes in the social worlds of the research site (Lempert, 2007). Memos are not intended to describe the social worlds of the researcher's data, instead they conceptualize the data in narrative form (Lempert, 2007).

Memo writing is essential to Grounded Theory methodological practices and principles and the fundamental process if researcher/ data engagement that results in a "grounded" theory (Lempert, 2007). Memo writing is the methodological link, the distillation process, through which the researcher transforms data into theory and the researcher analytically interprets data

through sorting, analysing, and coding the "raw" data in memos, the Grounded Theorist discovers emergent social patterns (Lempert, 2007). It is the methodological practice of memo writing that roots the researcher in the analyses of the data while simultaneously increasing the level of abstraction of his/her analytical ideas (Charmaz, 2006). Ultimately, it is the integration of these abstract analyses developed in the memos that the researcher shares with a public audience (Lempert, 2007).

4.6 Application of Grounded Theory

In order to analyse the selected data components, the principles of grounded theory are applied effectively as illustrated in Figure 4.1. There are five different phases to analysis of selected data components consisting of merging conceptual duplicates, primary classification, categorising classification, revisit change and renaming, and reviews. Central to analysis are grounded theory principles of coding, memoing, and constant comparison. The details of phases are elaborated in this section.

Merging conceptual duplicates – After the removal of textual duplication in data preparation phase section 4.4, the data components are dissected to understand and analyse the underlying meaning of the text. The words thought to be useful part of the concept were highlighted in bold. The details of the concepts were constantly compared with each other to find conceptual duplication present among data components. In Table 4.2 an example of merging conceptual duplicates is presented. The two data components numbered 18 and 20 discuss the similar underlying concept of guidelines and relevant to good documentation. The details of both data components are merged together. The different data components brought together in Table 4.2 are highlighted by using different colours of text.

Table 4.2: Example of merging conceptual duplication

Data Component No.	Data Component Before Merging of Conceptual Duplicates	Data Component After Merging of Conceptual Duplicates
18	guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options.	**Good documentation** invites people to interact with project open an issue or pull request. Use these interactions as opportunities to move them down the funnel (from users to contributors and to maintainers). **communication is public and accessible, anybody can read past archives to get up to speed and participate. Guidelines should at least provide details about the expeced code style, commit format, pull request process, and available communication options. Projects should provide a policy or comprehensive set of contribution guidelines. Write down your project's vision, a project roadmap,make those public as well, Add them to your README, or create a separate file called VISION. keep your documentation up-to-date. Be proactive to reduce the volume of unwanted contributions in the first place, explain your project's process for submitting and accepting contributions in your contributing guide. Fill out a issue or PR template/checklist and open an issue before submitting a PR**
20	Projects should provide a policy or comprehensive set of contribution guidelines	

Once this redundant activity of finding conceptual duplicates was complete, 11 data components were merged with the text of other data components. The complete master table containing 43 data components after the removal of 11 conceptual duplicates along with data sources appears in appendix B.2.

Primary classification – During primary classification the data components were grouped into categories on a preliminary level. Memos were written to identify and to record the rationale behind considering grouping of data components, under a particular category. A short few words description was written in the memo to identify emerging practices from the data components. An example of primary classification appears in the Table 4.3, where the details of the memo for the primary classification are recorded along with the preliminary category and a tagline for practice in "Primary Classification and Memo Description" column. The complete table with 43 data components with 2 practices added by the researcher explained in section 4.7, to benefit OSS projects (highlighted in yellow), are given in appendix B.3.

Table 4.3: Example of primary classification

Data Component No.	Data Components	Memo No:	Primary Classification and Memo Description
1	**Visualisation of Resources tool**- A visualization word cloud has been proposed **to show quickly the level of cooperation of the team in the project** [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors. // **Technology Oriented Tools**	1	Solution that invove technology to support knowledge relavant activites in OSS projects. All such practices that can be used as a rsolution to deal with knowledge loss due to contributor turnover are gathered under one **category "Technology oriented tools". Practice -> Utilising Technology Oriented tools**

Categorising classification - In categorising classification, data components now referred as practices are grouped under the main category. The main text to be considered for the phrasing of the practice is highlighted in bold appears under "data component" column. The main categories encapsulate and deliver the vital meaning of each category under which the data components are consolidated as practices. Overall eight categories are identified namely adoption of technology oriented tools, knowledge transfer and sharing, knowledge centralisation, encouraging knowledge contributions, uniform knowledge distribution, motivation, community health, and governance and leadership. Categories "encouraging knowledge contributions" and "motivation" also have sub-categories. An example of categories extraction from the contents of data components is shown in Table 4.4. The categorisation for all 45 practices are shown in appendix B.4.

Table 4.4: Example of categorising classification of practices

Data Component No.	Data Components (now visible as practices)	Memo No:	Categorisation of practices detailed in Memo
1	**Visualisation of Resources tool**- A visualization word cloud has been proposed **to show quickly the level of cooperation of the team in the project** [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors. // **Technology Oriented Tools**	1	Solution that involve technology to support knowledge relavant activites in OSS projects. All such practices that can be used as a rsolution to deal with knowledge loss due to contributor turnover are gathered under one **category "Technology oriented tools"-> Practice-> Visualisation of Resources tool to show quickly the level of cooperation of the team in the project**

Revisit change and renaming - At this stage consolidation of practices under main categories is complete and the researcher revisits the accuracy and consistency of classification. The description of practices has reduced in words and are more refined. In case of changes or renaming of the categories or practices, memos are written. The PKR practices were classified under 8 categories including sub-categories of "motivation" and "encouraging knowledge contributions". The category of "encouraging knowledge contributions" have 2 sub-categories consisting of "removal of knowledge sharing barriers", including 6 practices and "standardising project rules and policy", including 10 practices. The category of "motivation" has two sub-categories containing in all 6 practices, 3 on intrinsic motivation and remaining 3 on extrinsic motivation. In Table 4.5 motivation category with respective sub-categories can be viewed. The complete set of PKR practices before the reviews are presented in appendix B.5.

Table 4.5: PKR practices and relevant category

Proactive Knowledge Retention Practices Category	Proactive Knowledge Retention Practices
Motivation (6)	**Intrinsic Motivation**
	Appreciation written as newsletter or a blog post thanking contributors
	Recognition and appreciation for best answer provided
	Creating a culture to willingly share knowledge with other contributors
	Extrinsic motivation
	Rewarding contributor for knowledge sharing and transfer
	Using **gamification** qualities to evaluate who provides the best answer is the one that gains the most points
	Ascending meritocracy level of contributors on knowledge sharing and transfer activities similar to the one followed for code contributions on the project

Reviews – The PKR practices presented in entirety classified under categories, went through rigorous review process extending to eight different sessions. The reviewers consisted of the researcher and two supervisors with an expertise in relation to the application of grounded theory. The thought process followed during the review process was constantly recorded in memos. The different colours have been used during the review process to differentiate between changes made at every review (e.g. green for the first review, purple for the second review, amber for the third review and so on).

The **first review** emphasised on improving the naming convention, changing the perspective of practices by associating them with outcomes, refining the PKR, and attaining balance in the model i.e. more uniform structure of classification. The set of PKR practices in its current state had categories with three levels of classifications. Iterations helped to refine every practice in the current set of PKR practices and distinct practice are further refined under a category. The implication of associating process with the outcome, led to the emergence of more terms for the classification of practices. In all 14 terms were identified as categories while associating each practice with the outcome in the reference of its vitality to the OSS projects. An example of changes applied in first review appears in Table 4.6. The table for first review in entirety is given in appendix B.6.

Table 4.6: PKR practices the first review

Practice	**Outcome**	**Memos / Proactive Knowledge Retention Practice Category**
Adopting real-time visualisation of resources	Timely deployment of the resources on need basis with the help of the technology. Relocating knowledge workers based on technology. Timely deployment of the resources on need basis with the help of the technology.	Resources are managed in case of turnover using visualisation of resources. Tracking resources on the project and on need basis provide resources. Knowledge Resource Management
Keeping track of successor/ co-owner/ knowledgeable person/ with relevant expertise on the work of others	Reduces the risk of file/ code/ module abandonment.	Knowledge Resource Management

The **second review** concerns were raised on the systematic categorisation of the practices and readability with a focus on the process. To overcome the concerns in the second review, the emerging categories from the first reviews were processed further through constant comparison and the prominent categories were highlighted to accommodate all practices. Understanding the context of the emerging categories in the first review, the naming of categories is revised in the second review for clarity. At the end of second review, constant comparison resulted in 7 categories of KPR practices. The categorical structure of classification now attained uniformity and had simple hierarchy of only one level. To convey the meaning of the practice to its practitioners, the details of the practices were also revised to make longer text shorter and conversely the shorter text longer. An example of changes made is shown in Table 4.7. The table for second review in entirety is given in appendix B.7.

Table 4.7: PKR Practices the second review

KPR Practices	Memos	Proactive Knowledge Retention Practice Category
Adopting real-time visualisation of resources	Project leaders, owners or team leaders in OSS may relocate resources on projects based on the visualisation of resources when there are less resources than required. Decisions to use a technology tool that depicts the real-time visualisation of resources is mainly upon the main leaders of the project	**Governance and Leadership**
Keeping track of successor/ co-owner/ knowledgeable person/ with relevant expertise on the work of others	The team leaders would be keeping track and identifying people who are knowledgeable on the work of others. It can also be the culture/ Environmet on the project to keep track on the work of other contributors. A teamleader would have more resources to track people with similar area of knowledge while for an individual contributor it will be only known to him or her that who is working on code simialr to hers. OSS projects will benefit more if sucessors/ owners/ based on similar work history are tracked by the team leaders.	**Governance and Leadership**

In the **third review**, the first concern was on the usage of the model and the second on the distinction among different types of roles contributors exhibit as mentioned in the practices i.e. cores, non-cores, and integrators. Furthermore, rephrasing of the text constituting the practice name and description was required to refine and represent the set of PKR practices. The first concern was about using the PKR practices, which was resolved by adding a practical example on the application of every PKR practice and the second was rectified by documenting roles involved in OSS projects. An example of some rows highlighting changes is shown in Table 4.8. The table for third review in entirety is given in appendix B.8.

Table 4.8: PKR Practices the third review

Proactive Knowledge Retention Practice Category	KPR Practice	KPR Practice Description Rev 1 Rev 2 Rev 3	Memos
Communication	Establish communication mechanisms from cores to non-cores contributors	Knowledge communication from cores to non-cores Communicate high level ideas, suggestions, information, comments, instructions and answers to team members. Enagage in more code changes as core developers and through this share more knowledge with other team members. Knowledge communication from cores to non-cores	Knowledge communication between cores and non-cores is an important factor for knowledge trasnfer and sharing leading to uniform knowledge distribution The communication of the OSS project is directed by the core contributors. Their attitudes and involvement in knowledge sharing were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities. The more code changes core developers perform, the more knowledge they provide [124]. However, their least involvement in communication and task changes results into some negative team attitudes. Knowledge communication between cores and non-cores is an important factor for knowledge trasnfer and sharing leading to uniform knowledge distribution

During the **fourth review** it is realised that model contains some practices, which are more suitable to be considered examples of other practices. For instance, practice named "Job rotation to acquire different skills" is considered as an example of practice "Advocating diversification of core contributor's specialisation". Job rotation is for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices, such as pair programming,

facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation. Another practice on "adopting use of collaborative tool for code review" is removed from "governance and leadership" and is merged as an example of practice "establishing explicit code review guidelines", and details on the use of collaborative tool for code reviews are included in the description of the practice. An example of changes is shown in Table 4.9.

Table 4.9: PKR Practices the fourth review

Proactive Knowledge Retention Practice Category (32)	KPR Practice	KPR Practice Description Rev 1 Rev 2 Rev 3 Rev 4	Memo 4
Communication (5)	Establishing communication mechanisms from cores to non-cores contributors	Advocate active communication on matters such as high level ideas, suggestions, information, comments, instructions, answers to team members and also includes engaging with software code. The more changes made to the code, the more knowledge core contributors share with others specially non-core contributors. This practice should result in community of practice to create a network of experienced contributors to collect and exchange knowledge. For example, Siemens and BMW to resolve the challenges of losing expertise through retirement and attrition found a cross-company community network. Intel, Infineon Technologies, and Winterthur Insurance Switzerland joined the "Leaving Experts Community of Practice". Members of the community must be experienced in dealing with the problem of losing expertise and be able to contribute valuable lessons to member companies.	Changed the practice name from "Establishing Community of practice network". Rephrased the practice description. Added practice "Establishing Community of practice network" under knowledge sharing and transfer under this practice as an example.

Similarly, in the fourth review a thorough examination revealed that some practices with the evolution of model seem suitable in another category, e.g. "establishing explicit code review guidelines" is moved under "core development practices" category from "governance and leadership" category. The practice of "establishing explicit code review guidelines", after the internal review is

thought to be more suitable under "core development practices" category because it provides insights to code review guidelines.

The recursive process of constant comparison in the fourth review, to refine the model and re-organisation led to the change in the classification structure of practices and upholding them as examples of PKR practices resulted in obsoleting two categories namely "knowledge sharing and transfer" and "Removal of knowledge barriers". As a result, 32 PKR practices were classified under five categories. The table for fourth review in entirety is given in appendix B.9.

Table 4.10: PKR Practices the fifth review

Proactive Knowledge Retention Practice Category (32)	KPR Practice	KPR Practice Description Rev 1 Rev 2 Rev 3 Rev 4 Rev 5	Memo 5
Communication (6)	Advocating maximum limit of code review feedback time for all contributors	Due to the lack of an established reputation, peripheral developers or non-cores wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process. Accordingly, a delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor [135]. An improvement to the timings of the review feedbacks in OSS projects of peripherals or non-cores can result in a uniform distribution of knowledge, reduce the effects of turnover, and motivate newcomers to stay for a longer duration. This practice is included in communication category but can be categorised under "Core Development Practices" or under 'Motivation'. In this practice the emphasis is on providing timely feedback and minimising delay, which is more related to communication category rather than 'core development' or 'motivation' category. An example of overcoming delay to core review feedback time, is to set a maximum time limit to respond to a submitted code review by cores and non-cores alike.	**Revised the description of the practice for readability and understanding.**

The set of PKR practices during the **fifth review** did not require any changes in the classification scheme or the phrasing used for the practices. The remaining concern was to articulate a description of the practices that captured the real essence of the practice and to provide meaningful examples for its relevance to

OSS projects. An example of changes is shown in Table 4.10. The table for fourth review in entirety is given in appendix B.10.

The remaining three reviews were performed individually and comments were exchanged among researcher and the reviewers. The only focus now was to revise the evolved PKR canonical model and look for any remaining inadequacies and inconsistencies. The revisions in the last three reviews altered the description of some practices for clarity purpose with minor changes. As the PKR canonical model emerged with much clarity and more focused on the purpose, in the last review, the practice "advocate a maximum limit for code review feedback for all contributors" was realised to be more suitable as a guideline under the practice "establish explicit code review guidelines".

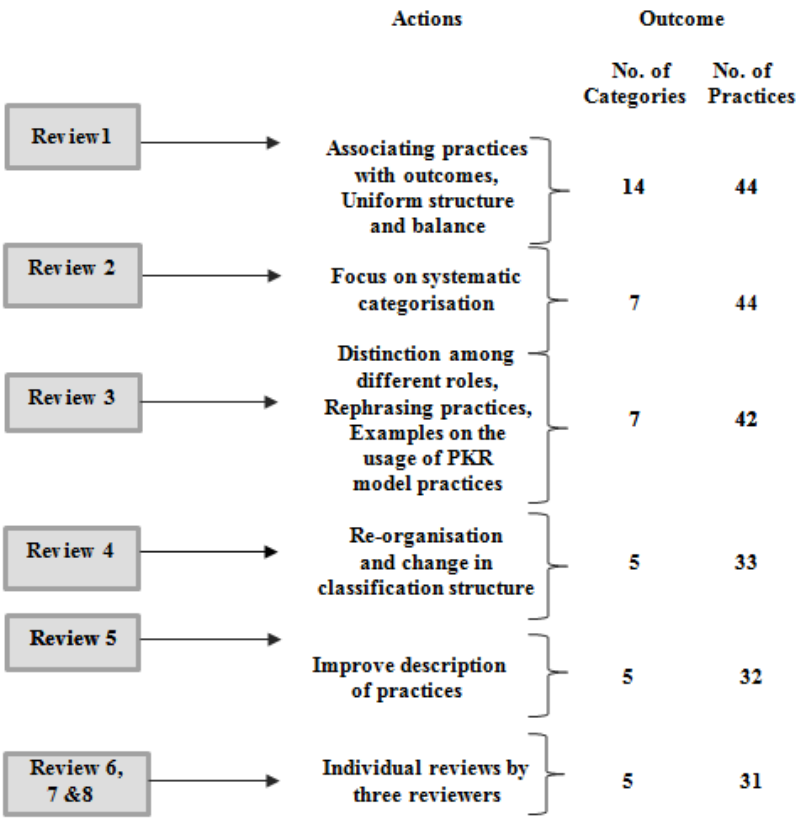

Figure 4.3: 8-step review process and outcome for every review with the number of categories and practices in PKR canonical model

The feedback from last three reviews on the PKR canonical model was effectively incorporated to further improve and refine the PKR canonical model

towards its final state with 5 categories and 31 practices. Figure 4.3 depicts the entire review process and summarises the outcome after every review conducted affecting the number of categories and number of practices in the PKR canonical model.

4.7 Practices by Researcher

There are three practices thought to be of benefit to OSS projects and were appended to the PKR canonical model by the researcher. Before the canonical model was presented in entirety, the researcher added two practices during primary classification: 1) Assessing contributors on the meritocracy level for knowledge sharing and transfer similar to the one followed for code contributions on the project, and 2) Project policy to invite contributors from all roles to participate in peer reviews to transfer project relevant knowledge and experience. The first practice suggests on introducing ascending meritocracy level of contributors on knowledge sharing and transfer activities similar to the one followed for code contributions on the project. The second practice focuses on proactive assignment of task to non-cores in OSS projects. This can also be an incentive to escalate meritocracy levels. The last practice that was added to the PKR canonical model is on projecting a policy that encourages peer review contributions from all project roles.

4.8 Canonical Model of Proactive Knowledge Retention in OSS projects

The PKR canonical model is composed of 31 practices grouped under five PKR categories. The details on each PKR category and practice are given in this section.

4.8.1 Communication

Knowledge is the product of an individual's experience and accumulates because of communication or inference (Zack, 1999b). On the completion of PKR canonical model five different practices are identified under communication category. The details of each practice with example(s) are given in the following passages. Note that P1 = Practice 1; P2 = Practice 2, etc.

P1. Establish communication mechanisms from cores to non-cores contributors

Communication between core and non-core contributors is an important factor for knowledge transfer and sharing leading to uniform knowledge distribution, as "communication in OSS projects is directed by the core contributors and their attitudes and involvement in knowledge sharing is linked to the demands of the wider project teams" (Licorish and MacDonell, 2014). Core contributors can bring higher level of skill and cognitive characteristics to their project teams. However, OSS teams with less communication from active core contributors can result in negative team attitudes. This kind of disruption in communication in OSS projects can hinder knowledge sharing. This practice advocates active communication from cores, while being the centre of their project's knowledge activities, on matters such as high-level ideas, suggestions, information, comments, instructions, answers to other contributor questions and also engaging with software code, as the more changes made to the code, the more knowledge they provide (Licorish and MacDonell, 2014) especially to non-core contributors.

This practice can be implemented through community of practice where a network of experienced contributors collect and exchange knowledge with non-core contributors. Community of practices are used as a long term strategy by management to retain knowledge before it is lost (De Long and Davenport, 2003). For example, Siemens and BMW to resolve the challenges of losing

expertise through retirement and attrition created a cross-company community network. Intel, Infineon Technologies, and Winterthur Insurance Switzerland joined the "Leaving Experts Community of Practice" (De Long and Davenport, 2003). Members of the community must be experienced in dealing with the problem of losing expertise and be able to contribute valuable lessons to member companies.

A community of practice can be employed in OSS project to create by involving experienced cores, who contribute towards sharing valuable lessons learned and experience on the project to non-cores. Another example of communication mechanism from cores to non-cores in OSS projects can be of using story telling to generate, share, and discuss stories for the quick integration of new learning. NASA, the World Bank and IBM have all made productive use of storytelling (De Long and Davenport, 2003). A third example to improve communication from core to non-cores can be of conducting interviews to capture knowledge and distribute it to other contributors by integrating it into an organisation. In OSS projects, interviews can serve to transfer knowledge from cores to non-cores. Accenture found the materials used to develop training content most likely come from face-to-face interviews conducted before an employee leaves the organisation (De Long and Davenport, 2003).

P2. Explicit identification of communication protocol/ mechanisms between integrators and contributors

Communication is important to the progress and success of the project, contributors need to understand what protocol, and mechanisms exist for communications in a project. In a typical OSS project the maintainer role has commit rights on the project, specifically the main branch. The contributor wishing to make modification creates a project's branch and after completing changes, sends a request to the project 'maintainers', who has commit rights to the project. Furthermore, an "integrator", merges a branch (a subset of the

same project) into the main branch of the project. Integrators should be proactive by establishing a professional communication etiquette, and reactive by following discussions and intervening in cases where discussion diverges from the etiquette. For instance, integrator and contributors should agree on minimum communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange (Eghbal et al., 2018). The communications mechanisms such as the use real-time communication channels (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub) (Eghbal et al., 2018). Furthermore, communication is public and accessible; anybody can read past archives to get up to speed and participate.

P3. Establish response time parameters for project queries or issues

This practice focuses on the timely response to queries or issues on the project, which helps to engage contributors and make them feel more involved in the project. Furthermore, interest of contributors looking forward to get involved in the project will be heightened when they receive timely response to their queries resulting in new contributors on the project. Responding effectively when someone files an issue, submits a pull request, or asks a question about the project, makes the project community feel that they are part of a dialogue and builds their enthusiasm to participate. In case, the request cannot be reviewed immediately, acknowledging it early helps increase engagement among project community. For example, setting up notifications on social media or channels i.e. Stack Overflow, Twitter, or Reddit issue an alert when someone mentions a project, an alert is generated to the responsible person (Eghbal et al., 2018). This mechanism can be employed to help establishing response time parameters in OSS projects.

P4. Actively elicit views of project community on major concerns

Eliciting the views of the wider project community is important. Making other people feel heard, and committing to resolving their concerns, goes a long way to diffuse sensitive situations. Ensuring that everybody feels heard and

that all information has been made public before moving toward a resolution. This practice adheres more to achieving communication objectives by focusing more on the adopting an approach to reach out to contributors and find out their views on major concerns through open discussions. For example to elicit views of project contributors, the emphasis is on "consensus seeking" rather than consensus (Eghbal et al., 2018). Community members discuss major concerns until they feel they have been adequately heard. On the contrary, voting emphasises getting to an "answer", rather than listening to and addressing each other's concerns.

P5. Promote documentation of solutions to problems commonly experienced by multiple users

OSS users can face many issues while working on a project. This practice promotes the documentation of common solutions to problems experienced by users and access to all contributors on the project (Eghbal et al., 2018). This practice demonstrates the use of documentation as an effective mode to communicate knowledge to the contributors of the project. An example of implementing this practice can be when multiple users run into the same problem, document the answers in the some file such as README and provide access to it to all contributors.

4.8.2 Contributor Motivation

P6. Formalise a knowledge contribution recognition program

In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the project. A reward structure can be based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition, whilst extrinsic motivator is related to monetary incentives (Gammelgaard 2007). Reward is associated with extrinsic motivation of a contributor and relates to encouraging knowledge

providers by rewarding them. To encourage contribution of knowledge, based on codification (formal documentation) and personalisation (sharing knowledge through socialisation) a reward system is established for people documenting and sharing knowledge (Hansen et al., 1999) and rewarding knowledge provider's by recognition. Organisations like Xerox, and Hewlett-Packard reward people for sharing their knowledge (Rus et al., 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer, 2003). Similarly, a knowledge recognition program can be organised in OSS projects, where project contributors can learn from other community members and apply knowledge in their projects. A recognition program can be associated with knowledge activities comprising of sharing, documenting, disseminating, and any other action facilitating knowledge propagation.

P7. Reward active knowledge contributors with project seniority

Contributors progress to become a core contributor in OSS project based on meritocracy and mainly on the number of the code contributions they submit. The practice suggests to utilise the concept of extrinsic motivation and have an additional criteria for assessing a contributor based on knowledge sharing and transfer activities on the project. Active knowledge sharing can be seen as a mechanism to contribute towards contributor's seniority. In particular a gamified environment has important implications for KM in software engineering (Vasilescu et al., 2014) and OSS projects, as gamification has the potential to increase engagement of knowledge providers.

There is also intrinsic motivation for the knowledge provider such as altruism or learning by helping others solve problem (Vasilescu et al., 2014). Contributors in OSS communities may earn high status because of their free knowledge sharing activities (Wasko et al., 2005). Status can place contributors in high demand and can bring opportunities in the future for securing monetary rewards. The reward

is not formal or assured, but may be a motivator to participate.

For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted (Zagalsky et al., 2016). In Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points (Zagalsky et al., 2016). Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information (Zagalsky et al., 2016). Here the number of points gained by the contributor can serve as a extrinsic motivation to gain recognition and earn a repute in OSS community. At the same time gamification identifies people who are more knowledgeable and active based on the point given to them.

P8. Promote a culture of appreciation for knowledge contributors

Appreciation is an extrinsic type of motivation for contributors. This practice encourages the cultivation of a culture in OSS projects, where the objective is to motivate contributors to share knowledge by appreciating their efforts in the project. An example of this practice can be similar to thanking contributors using newsletters and blog posts (Eghbal et al., 2018) that visibly and openly appreciate their constant efforts to share knowledge and support make project a success.

4.8.3 Core Development Practice

P9. Encourage pair programming and shared code ownership

In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested (Mens, 2016). Pair programming facilitates knowledge sharing between peers and is considered under the category of core development practice as it represents a

proactive approach towards KR in OSS projects. It further can facilitate effective knowledge sharing in the OSS projects.

Examples of pair programming discussed here are remote pair programming and a pairing variation namely expert novice pairing. Remote pair programming is not just for the corporate sector and can also be implemented in open-source projects (Monus, 2018). It was found that effective remote teaming could be done with the PC sharing software and audio support Baheti et al. (2002). Pair programming only works remotely when both developers can see the screen where the code is written and the key is to find the right screen sharing software for you and your colleague (Monus, 2017).

Expert-novice pairing creates many opportunities for the expert to mentor the novice. This pairing can also introduce new ideas, as the novice is more likely to question established practices. The expert, now required to explain established practices, is also more likely to question them. However, in this pairing, an intimidated novice may passively "watch the master" and hesitate to participate meaningfully. Also, some experts may not have the patience needed to allow constructive novice participation (Williams and Kessler, 2002).

P10. Project a policy that encourages peer review contributions from all project roles

Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors (Rigby et al., 2014). Peer review is a collaborative activity and inviting contributors from different roles can lead to uniform knowledge distribution from cores to non-cores. This is included as the core development practice since peer review is one of the key development activities in OSS development. An example of this practice can be that core team proactively invites contributors irrespective of their role on the project and encourage them to participate in peer reviews.

P11. Classify defects for suitability across the various contributor levels

Labelling a bug to suit the level of contributor expertise will benefit OSS project to recruit more contributors and get them started with tasks according to their knowledge and expertise level. In addition, labelling bugs helps in identifying the types of tasks and facilitate their resolution based on the suitability of contributor. This practice is included under the category of "core development practices", because it benefits the development of the OSS project. For example, "first timers only", "good first issue", or "documentation" are examples of labels make it easy for someone new on the project to quickly scan issues on the project and get started (Eghbal et al., 2018). Similarly, resisting fixing easy (non-critical) bugs by core contributor provides opportunities to recruit new contributors.

P12. Ensure the presence of test files and associated testing artefacts

Test files and testing artefacts help in testing the software code and can be one of the core development practices to directly impact the quality of the code. Therefore, it is of utmost importance to ensure the presence of test files and associated testing artefacts. Test files include tests and other checks to improve

and ensure the quality of the software code. A contributor in the OSS project can effectively utilise test files and artifacts to test her code for quality purpose. This practice facilitates programmers at any level from novices to experts to test their code for bugs, fulfilling the requirements, and invalid outcomes before sending it off for review. For example, if tests are added for the project, explanation should be provided on how they work in a file specifically designated for this purpose e.g. Test Files (Eghbal et al., 2018).

P13. Label substantial new items of work "Work In Progress"

Labelling on going work as "work in progress" attracts more contributors interested in the ongoing updates. Furthermore, more expertise in the OSS development is pooled in by such informative labelling activity. It can be a piece of information on current work that is communicated to contributors. This practice encourages and invites more contributions while working on a substantial update, and therefore is included in the category of "core development practices". For example, this practice can be applied, while working on a substantial project update, through a pull request and marking it as a Work In Progress (WIP). That way, other people can feel involved in the process early on (Eghbal et al., 2018).

P14. Introduce non-restrictive commit access to contributors where appropriate

In OSS project not every contributor has right to commit code. The code review policies describe the process for commit access, where generally code commit access is given only to selected contributors. This practice advocates non-restrictive access to contributors (Eghbal et al., 2018). The practice can be applied by giving contributors commit access on the project where appropriate, to allow them to be more excited to polish their patches. This expedites the learning and correction process. Further benefit of applying this practice can be that it helps in finding new maintainers for projects that had not been worked on in a while. An example of this practice can be to allow contributors to

commit their code to a repository where it does not effect the main branch, but at the same time contributor code is reviewed by other contributors.

P15. Establish explicit code review guidelines

Code review guidelines are effective in the review of code and associated corrections. Code review checklists can be a useful way to provide team members with clear expectations for each type of review. The use of a collaborative code review tool is recommended to allow reviewers to log bugs, discuss them with the author, and approve changes in the code guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options.

One example of code review guidelines can be to advocate a maximum limit for code review feedback for all contributors. The main focus is to respond to a submitted code review by cores and non-cores alike. In particular, quick review feedback from core to non-core contributors is positively associated with knowledge preservation and exchange. It has been observed that due to the lack of an established reputation, peripheral (or non-cores) developers can wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process ((Bosu and Carver, 2014). Accordingly, a delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor (Bosu and Carver, 2014). An improvement to the timings of the review feedbacks in OSS projects of peripherals or non-cores can result in a uniform distribution of knowledge, reduce the effects of contributor turnover, and motivate newcomers to stay for a longer duration. An example of overcoming delay to core review feedback time, is to set a maximum time limit to respond to a submitted code review by cores and non-cores alike.

P16. Release post-mortem reviews to all contributors

Post mortem or after-action review promotes learning through a collective activity to build knowledge based on the experience and to improve future practice. In post-mortem reviews, learning takes place through socialisation and

when individuals share experiences, tacit knowledge is externalised, thus knowledge is shared from individual level to organisational level. Post-mortem reviews are an attempt to codify knowledge from projects, where the main output is the report, and can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects.

For example, Apple used a method for post-mortem approach (Collier et al., 1996) based on a project survey, collecting useful information on project, a debriefing meeting, and a 'project history day'. Microsoft invests considerable efforts into writing 'Post-mortem reports' (Dingsøyr, 2005). The reports contain details on what has worked well for the last project, what has not worked well and teams that need improvement for the next project (Cusumano and Selby, 1998). In OSS projects post-mortem reports can be maintained to codify the experiences and lesson learned on the project and improve the future projects.

4.8.4 Environment/ Ecosystem/ Culture

P17. Build a supportive community for conflict resolution

"Any popular project will inevitably attract people who will deliberately or otherwise harm, rather than help project's community and this may start unnecessary debates, quibble over trivial features and escalate to potential bullying of others" (Eghbal et al., 2018). If left unchecked, negative people will make other people in community uncomfortable. Project leaders should adopt a zero-tolerance policy towards these types of people. An OSS community with an ability to resolve conflicts will last longer and with time evolve and more productive. For example, adopting a code of conduct builds a supportive community is the key to resolving conflicts. "A code of conduct is a document that establishes expectations for behaviour for project's participants. Adopting, and enforcing, a code of conduct can help create a positive social atmosphere for project community" (Eghbal et al., 2018).

P18. Explicitly identify project contributors

The contributors who contribute to the project can be listed along with other details of the project. This practice can facilitate to maintain the network of contributors in OSS project and helps to establish an ecosystem that can be a source of linking contributors in OSS community. Explicit identification of contributors who contribute to the project can also be helpful for other to reach out to the contributors on the project for queries, solutions, and support. As a result, more contributors will end up joining the project. Example of identifying the contributors can be that designating leaders simply as add their names to README or a CONTRIBUTORS text file. CONTRIBUTORS or AUTHORS file in the project that lists everyone who contributed or contributes to the project (Eghbal et al., 2018).

P19. Allow self-organization of project roles

Self-organisation is related to letting people decide what role they want on the project. In OSS project, contributors when allowed to choose roles by their discretion gives them a sense of freedom and independence to operate in a dispersed and distributed community synced mostly through asynchronous means of communication. This practice will also create an environment where contributors are more cooperative and respectful towards each other's opinions and suggestions, while collaborating on different tasks. This practice can also help in the propagation of sharing knowledge among contributors since they are more enthusiastic about their roles and willing to elaborate on the tasks they are performing to others. For example, letting people self-organize and volunteer for the roles they are most excited about and interested in, rather than assigning them (Eghbal et al., 2018).

P20. Create a culture to share knowledge altruistically

Creating knowledge sharing awareness among contributors and evolving an altruistic philosophy is a primary motivation behind many OSS project. Knowledge sharing should be introduced as a vital cultural element in OSS

projects in order to stabilise ecosystem and extending selfless gesture of doing good to the society. This practice introduces the idea of sharing knowledge as a part of a core culture where contributors willingly share knowledge as a selfless deed and for the betterment of OSS community. As an example of intrinsic motivation is, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated (Lindvall and Rus, 2003).

P21. Make project documentation publicly accessible

Making project documentation publicly accessible enhances the visibility of a project. This practice also enables the building and strengthening of the ecosystem, as the project and project document should have an open access to all project community. When communication is public and accessible, anybody can read past archives to get up to speed by going though mailing lists, blogs and participate. This can be related to the communicating changes in the documentation on the fly, but it is more of an adoption of a culture or environment where documentation is given public access to be viewed by the contributors of the project. For example, public communication can be as simple as directing people to open an issue instead of emailing directly to project leader or commenting on project's blog. Further, setting up a mailing list, or creating a Twitter account, Slack, or IRC channel for people to talk about the project (Eghbal et al., 2018).

P22. Foster an open-minded culture towards diverse types of contributions

OSS project governing teams and other contributors should adopt a culture to be open and accepting towards all kind of contributions, no matter how small or large it is. This kind of openness and culture encourages contributors who are casually contributing to the project but their efforts are recognised and accepted. For example, the types of contributions to accept can be a bug report or a small

fix making it easier for casual contributors to contribute (Eghbal et al., 2018).

P23. Encourage mentorship of new comers

Mentoring new contributors helps promote enthusiasm and a willingness to work on the project and submit quality contributions. Mentoring is suggested to be a logical approach for transferring important tacit and implicit knowledge (De Long and Davenport, 2003). It is argued that most companies due to resource limitations and an inadequate availability of experts find mentoring difficult to sustain in long term (De Long and Davenport, 2003). This practice is useful to overcome knowledge barriers faced by OSS contributors who are new to a project and are also enthusiastic about the project, but need a bit assistance in skills or knowledge can be considered for mentoring through their first contribution. For example, resist fixing easy (non-critical) bugs. Instead, use them as an opportunity to recruit and mentor new contributors (Eghbal et al., 2018).

4.8.5 Governance and Leadership

P24. Update project documents frequently

This practice is useful for the removal of knowledge barriers and is the responsibility of the contributor working on the project to ensure that documents are up to date. In case of problem or clarification required on the process it should be brought to the attention of team leaders. This practices advocates accessibility and correctness of knowledge available. The knowledge centralisation and updating has to be monitored by team leaders and its accessibility to contributors and teams. Also team leads or people governing OSS project are generally more knowledgeable to update the documents at the central location. For example centralisation as explained while inspecting the governance structures in Ericsson, which is argued to be similar to OSS, and where a centralised approach is implemented to secure quality (Britto et al., 2016). The knowledge loss was due to the recurrent movement of resources in

and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams. Centralisation also involves actively maintaining the project documents.

P25. Enable a strategy of successor identification

Identification of successors and involving other contributors as co-owners with relevant expertise knowledgeable on the work of other contributors is presented as a method to reduce the risk associated with developer turnover (Rigby et al., 2016). The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks (Rigby et al., 2016). A visualization word cloud has been proposed to show quickly the level of cooperation of the team in the project (Fronza et al., 2013). Intensity of colour and size of the letter in a wordle indicate the need for resources. In a similar fashion, adopting real-time visualisation of resources can facilitate in the identification of successors. Project leaders, owners or team leaders in OSS may relocate resources on projects based on the visualisation of resources when there are less resources than required. Decisions to use a technology tool that depicts the real-time visualisation of resources is mainly upon the main leaders of the project.

Another example is to develop a Transactive Memory Systems / Knowledge map / Knowledge portals. An insight into the area of expertise members, a knowledge map or directory can be used on the project website (Chen et al., 2013). In order to leverage OSS project teams, a focus is required towards facilitating the TMS development within the teams, based on knowledge location, the usage of the developer mailing list and knowledge credibility (Chen et al., 2013). The team leaders would be able to keep the track and identifying people who are knowledgeable on the work of others. Transactive Memory System development within the teams is based on knowledge location, the usage of the developer mailing list and knowledge credibility. An insight into the area

of expertise members, a knowledge map or directory can be used on the project website internal knowledge portal, serving as a way to help knowledge workers familiarize themselves with their colleagues knowledge, K- Portals are rapidly evolving into broad-based platforms for supporting a wide range of Knowledge worker tasks.

P26. Proactive assignment of varying tasks to non-cores

The skill set of core contributors is generally better than that of non-core contributors. Typically cores contributors make 80% of the knowledge contributions in OSS projects. In order to uniformly distribute knowledge from cores to non-cores, the team leads can proactively assign different tasks to non-core contributors. Leaders in OSS projects can govern the proactive assignment of different tasks to non-cores. As indicated that contributors who modify code from other contributors stay longer on the project (Lin et al., 2017). This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.

P27. Advocate diversification of core contributor specialisation

For an OSS project to survive, a diversity of core developers is required (Wahyudin et al., 2007). When a key contributor abandons an OSS project it can often reveal a very fluctuating proportion of developer contribution. An imbalance can occur between the contribution submitted and response from specialised core developers in community due to missing diversification. Forming a "core team" of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct), as suggested can be the solution to the imbalance in diversification of cores (Eghbal et al., 2018).

As an example, job rotation allows for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse

knowledge domains (Fægri et al., 2010). Some development practices, such as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation (Lindvall and Rus, 2003).

P28. Distribute project leadership

Distributed project leadership is related to sharing project ownership. This practice helps community to find support in the absence of the main leader and to keep the project maintenance ongoing. Maintenance in OSS projects continues the evolution process of the developing system. Shared ownership facilitates the project and the maintainer to either step away from project, on hiatus or permanently, request other contributors to take over. For example, a project leader should find support for project users and community while she plans to be away from a project and be sure to communicate her unavailability and inform contributors that who the new or temporary leader is (Eghbal et al., 2018).

P29. Explicit process for role progression The policy to progress in the project should be elicited by the team leaders on the project and made explicit for all project contributors. It is important for all contributors to see what can be the outcome of their efforts reflected as their personal progress on the project. This also serves as a motivation to contribute in order to progress to the highest level of recognition on the project. For example, documenting the process on the project to progress down the funnel i.e. from users to contributors and to maintainers (Eghbal et al., 2018).

P30. Document project rules and policies

Document project specific rules, policies and roadmaps should be provided by the project leaders. There can be changes made later to the rules and policies but the responsibility of documentation mainly lies with the leadership. Also for a person joining a new project, provision of these documents enables them to evaluate on the basis of the elaborated rules and policies. Transparency about project's roadmap, the types of contributions required for the project,

how contributions are reviewed, or why certain decisions are made on the project. Clearly state rules, policies, project vision, goals and any related information in a dedicated file such as Readme that is accessible to project contributors (Eghbal et al., 2018).

P31. Establish mechanisms for training

Training is an effective mechanism for knowledge transfer. Training of contributors is important in overcoming potential knowledge barriers in OSS projects. Removal of knowledge barriers: namely, lack of technical experience, lack of domain expertise and lack of knowledge on project hinder contributions. For example training is also a knowledge transfer practice found to include some combination of formal classroom training, e-Learning, video or computer-based training, on-the-job training, coaching, and shadowing (De Long and Davenport, 2003).

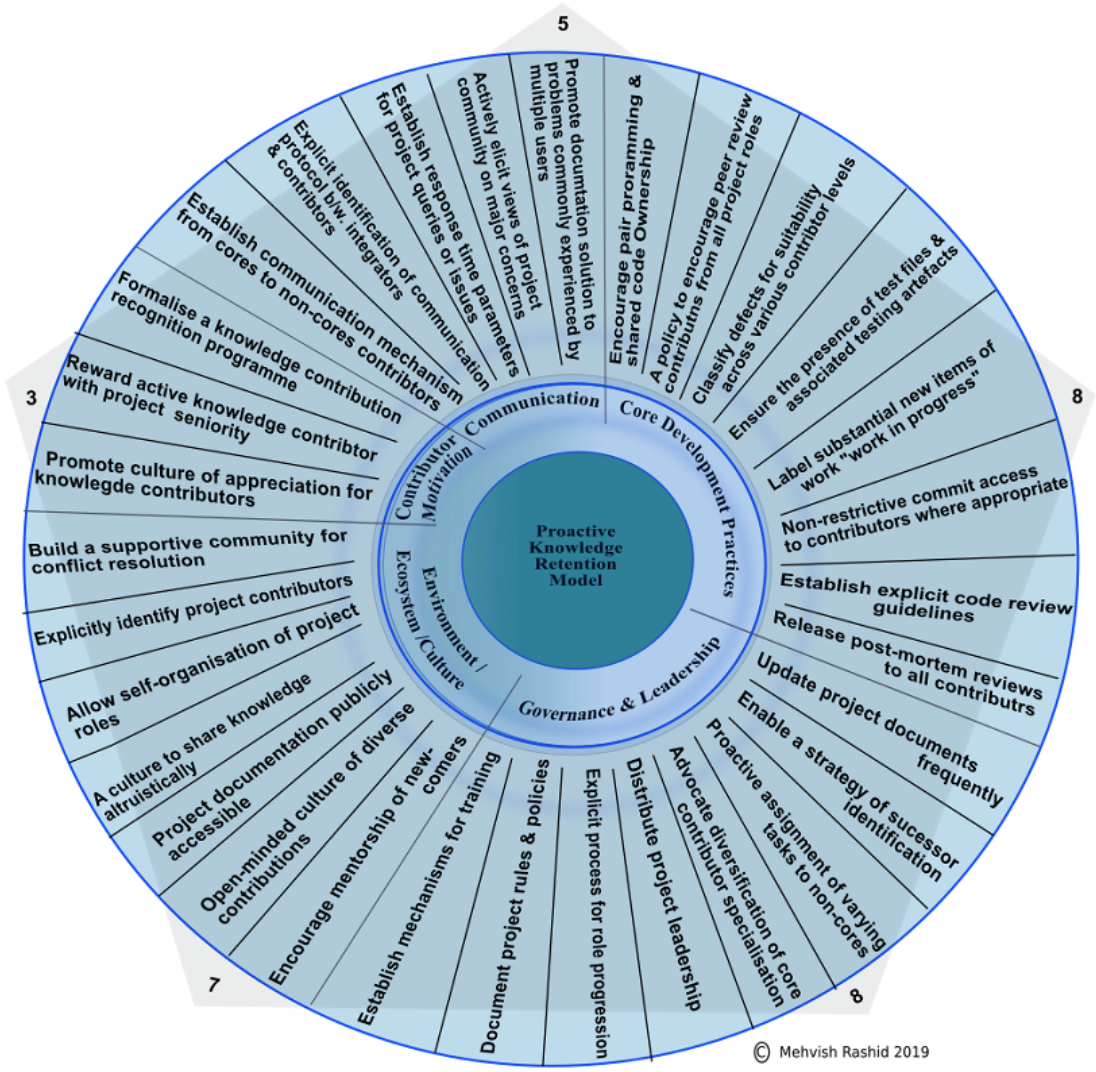

Figure 4.4: Proactive Knowledge Retention (PKR) model developed for OSS projects

4.9 Chapter Summary

This chapter articulates discussion on the rigorous process followed to develop the PKR canonical model and concludes with the identification of 31 practices classified under five categories. The description of each practice is provided in detail along with example from the real world and its utility in OSS projects. The overall structure of the practices is illustrated in Figure 4.4. A number on every edge of the pentagon indicates the number of practices identified under each category. The next chapter 5 discusses the development of the survey

instrument and its deployment for data collection to evaluate the effectiveness of PKR canonical model based on the feedback from OSS contributors.

Chapter 5

Survey and Data Collection

This chapter discusses the development of the survey instrument by strictly adhering to survey development process from the literature with some adaptation as presented in the following section. The developed survey instrument is further deployed for the data collection, which is described in the later part of this chapter.

5.1 Survey Development Process

Survey is thought to be more than an instrument in the form of questionnaire or checklist to gather information (Kitchenham and Pfleeger, 2008). Accordingly, survey is considered a comprehensive research method to collect information and utilise it in describing, comparing or explaining knowledge, attitudes and behaviour (Fink, 2003). In this work, the survey process is adapted from the following six main activities iterated here (Kitchenham and Pfleeger, 2008):

- Setting the objectives
- Survey design
- Developing the survey instrument (i.e. the questionnaire)
- Evaluating the survey instrument
- Obtaining valid data
- Analysing the data

The survey process includes the steps above and an additional seventh step of survey reporting (Ciolkowski et al., 2003). Survey reporting records the execution of the survey and reports the survey results (Kitchenham and Pfleeger, 2008). The following sections further detail each activity involved in the survey development process. Figure 5.1 depicts the process of survey development.

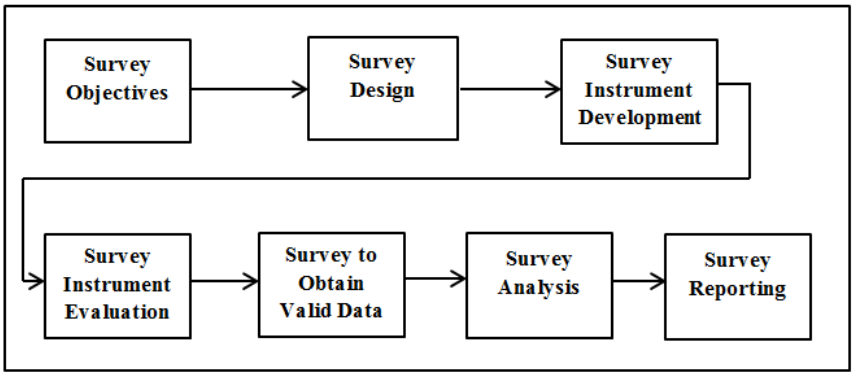

Figure 5.1: The process of survey development

5.2 Setting the Objectives

OSS projects might be considered to represent a plethora of accumulated knowledge in the form of contributors or knowledge workers with varying levels of experience and expertise. Contributions themselves represent one aspect of knowledge sharing in OSS communities. Contributions are of various types including bug reporting, bug fixing, documenting, code writing, participating in code reviews, and replying on mailing lists to resolve problems faced by other contributors. Due to the transient nature of contributors in OSS projects causing KL, the project-relevant knowledge at times remains unshared or becomes inaccessible to others. The author of this work suggests that KL in OSS projects due to contributor turnover can be reduced by introducing PKR practices in OSS projects that can transform contributors' use of knowledge and engagement in knowledge relevant activities including knowledge sharing and knowledge transfer. "The heart of any knowledge-retention strategy is its knowledge-sharing practices" (De Long and Davenport 2003).

The goal of this survey is to gather data on OSS projects and gain an understanding of contributors and feedback on PKR practices to support the research objective of systematically constructing a robust canonical model for OSS project PKR and through engagement with practitioners.

5.3 Survey design

The design of this survey incorporates the principles of scientific rigor. The dataset created in this survey will be obtained through engagement with the open source community. The privacy and safety of respondents will be prioritised, while maintaining the scientific rigor of the survey design and fielding process. This survey design considers the following activities to operationalise survey goals consisting of important decisions and activities as follows (Ciolkowski et al., 2003):

1. Defining the target population and the survey sample
2. Derive a conceptual model of the objects and variables of the survey
3. Approach for data collection
4. Survey instrument design
5. Approaches for data analysis
6. Validity issues

5.3.1 Defining Target Population and Survey Sample

GitHub is the world's largest collection of OSS (Borges et al., 2016), with 85 million repositories hosted and supporting a community of 28 million people to learn, to build and to share software, as reported by the GitHub website on June 2018 [1] . The users of OSS projects on GitHub use stars to indicate their interest and likeness for the project, which serves as a proxy measure of their popularity

[1] https://github.com/about

(Borges et al., 2016). The target population for this survey is the open source community. In order to fulfil the major goal of this survey, which is to evaluate a canonical model of PKR in OSS projects, a sample comprising of OSS projects would suffice for evaluating the proposed canonical model.

The projects will be selected based on popularity from a collection of GitHub OSS projects based on the number of stars as a proxy for popularity because it reveals how many people manifest interest or appreciation to the project (Borges et al., 2016). Another study limits the number of repositories to the top 5,000, based on the number of stars, to focus on the maintenance challenges faced by highly popular projects (Coelho and Valente, 2017). The sampling of OSS projects from popular OSS projects listed in GitHub will follow an approach similar to (Coelho and Valente, 2017). The details on the selection of OSS projects are given in the later section 5.8.2.

5.3.2 Conceptual Model of Survey

The conceptual model for this survey follows from Goal, Question, and Metrics (GQM) approach. GQM states the goals in advance leading to a selection of only those metrics that are relevant for achieving these goals (Koziolek, 2008). After defining the goals, the questions are asked to attain the goals, and data is collected to answer the questions. There are two areas identified for data collection: contributor profile, and evaluation of canonical model for PKR in OSS projects. The two areas for data collection are selected considering their relevance to the goal of this survey.

The set of contributor profile questions are designed to enable a deeper evaluation of the data obtained, for example based on the experience of respondent,or on the OSS role type of respondents. classify them based on their contribution in OSS projects. The second and more substantial part of the survey is concerned with the evaluation of PKR practices. The categorical difference among contributors based on their experience serves as a

distinguishing factor for the evaluation of PKR practices. The feedback and evaluation on PKR practices gives an insight into the respondents' understanding of the value of individual practices. These two survey components, along with their dimensions, are presented as a conceptual model for this survey in Table 5.1.

Table 5.1: Conceptual model of survey

Areas	**Dimensions**
Contributor Profile	Contributor's Experience Contributor's Role
Impact of leaving contributors	Primary impacts identified from literature
Canonical Model for Proactive Knowledge Retention	Canonical model of practices, obtained from: • Literature Review in OSS • Traditional Organisations • Open Source Guide

5.3.3 Data Collection Approach

Surveys are useful in collecting qualitative, quantitative or both types of data through questionnaires or interviews (Wohlin et al., 2003). The data collection is through a self-administered questionnaire (Kitchenham and Pfleeger 2008), which is designed using Google Forms. The data collection is anonymous and to manage sending surveys and set up follow-ups with partipants, GMass [2] is utilised. More details on data collection are elaborated in section 5.8.4. Anonymity of participant is preserved as a means to promote honest and uninhibited feedback from OSS contributors.

5.3.4 Survey Instrument Design

The design of survey instrument in this work embeds a series of questions in the form of a questionnaire. Interviews and questionnaires focus on asking a series of questions based on one or two types of questions: open questions and closed

[2]https://www.gmass.co/

questions (Lethbridge et al., 2005). A question is open when the respondents are asked to frame their own reply and a question is closed when the respondents are asked to select an answer from a list of predefined choices (Kitchenham and Pfleeger, 2008). Likert scales and multiple-choice questions are considered to be examples of closed questions and conversational responses are open questions with a textual input. It is recommended to always have some open questions to gain information that cannot be conveyed by more specific information seeking questions (Lethbridge et al., 2005). The design of survey instrument in this work considers both open ended and close ended questions.

5.3.5 Approaches for Data Analysis

The approach to analyse data depends on the type of measurement scales used. Classical measurement theory defines four basic types of measurement scale (Ghiselli et al., 1981):

1. Nominal: The scale values are unordered categories, and no mathematical manipulation makes sense.

2. Ordinal: The scale values are ordered, but the intervals between the values are not necessarily of the same size, so only order-preserving manipulations such as ranking make sense.

3. Interval: The scale values are ordered and have equal intervals, but there is no zero point, so only sums and differences make sense.

4. Ratio: The scale values are ordered and have equal intervals with a zero point, so any mathematical manipulation makes sense.

The responses obtained in this survey consist of a single selection from the list (nominal), multiple answers from the list (nominal), selection on Likert scale (ordinal), and text entered as passage. During data analysis, the selection of method is such to align with the type of data obtained and the objective of

research. The approach adopted in this work for data analysis is entailed in chapter 6.

5.3.6 Validity Considerations

Validity issues forecast any possible problems with survey design during survey implementation and execution (Ciolkowski et al., 2003). Validity considerations for this survey include internal validity, external validity, experimental validity and construct validity as described here:

- **Internal validity -** The control in a survey is usually quite low and it is impossible to know whether the respondents answer truthfully, or whether other effects bias the results (e.g., history effects—external events that influence someone's view and attitude towards a question) (Ciolkowski et al., 2003). In order to promote internal validity and allow the researcher to draw conclusions, a duration for respondents to take this survey is set at 9 weeks. Internal validity is also concerned with possible inaccurate and imprecise responses (Ciolkowski et al., 2003). This survey minimizes this error by carefully designing the questionnaire and by conducting a pilot study (Ciolkowski et al., 2003).

- **External validity -** deals with the extent to which the results represent the population. The sample of OSS projects collected from GitHub is based on the initial selection of 1020 different projects (one repository is equivalent to one project) representing the target population. While considering the number of contributors from the top 1020 OSS projects, there are too many contributors to survey, and the amount of effort required for such is simply too large. Therefore, obtaining a large number of respondents in absolute terms is considered a pragmatic approach (i.e. 1020 projects include more than 25000 contributors and this is a large population from which certain useful inferences can be made). The details on the selection of targeted

population and OSS contributors appear in the later section 5.8.1.

- **Experimental validity -** relates to the reproducibility of results with a different sample. In this survey, the reproducibility of results may vary from the same respondents over a given time span, as their experience and views evolve.

- **Construct validity -** relates to asking the right questions in the survey instrument to evaluate object of interest. In this survey, the questions are derived from the conceptual model (Section 5.3.2) and a pilot study further minimises the construct validity threat.

5.4 Survey Instrument Development

The survey instrument design includes one open question, and multiple closed questions. There are questions with pre-defined answers and with an additional response of 'other' as appropriate, to obtain a participant's opinion and to enable the elicitation of a richer and more informative suite of responses from participants. The survey also uses 11-point Likert scales, which has an equal percentage of positive and negative choices against a question, with an additional option of selecting neutral options (Allen and Seaman, 2007).

The operationalisation of survey goals (Ciolkowski et al., 2003) requires inspection of a conceptual model and its transformation into a set of questions. The survey goals are operationalised in the survey instrument for this research by associating them with a set of questions grouped together under two different sections namely, project contributor profile and feedback on PKR practices.

Contributor profile - The profiling of contributors in OSS projects provides the details on their total experience in OSS projects, role, and their overall experience in programming. There are four profile questions in the survey that enabled the extraction of rich data from the evaluation of the PKR practices.

Evaluation of PKR practices - A canonical model developed in chapter 4 represents PKR practices in OSS projects. The canonical model is evaluated and finalised through widespread and systematic practitioner engagement. The data components utilised to structure the canonical model are gathered from: the knowledge sharing and transfer practices mentioned in OSS literature; knowledge retention and transfer practices from traditional organisations in literature; and the open source guide hosted online [3]. The open source guide provides community-led input on the conduct of open source communities but is not a published artefact that is routinely referred to or adopted in published academic literature. The evaluation of KR practices facilitate in evaluating the draft canonical model, ultimately producing a robust canonical model for PKR in OSS projects that is firmly rooted in the literature and in the practitioner expertise and opinion.

Surveys instruments designed by other researchers, while studying OSS communities, used as a reference for some of the questions in designing this survey instrument are listed here:

- GitHub in 2017 [4] : A detailed survey designed to collect data about the attitudes, experiences, and backgrounds of those who use, build, and maintain OSS (Geiger 2017).

- R community [5] : Survey designed to study the information seeking and information providing behaviours in R community (Vasilescu et al. 2014).

- Pull-based Request [6]: Survey designed to understand the contributor's perspective on the GitHub pull based model to guide the design and process of the tools to reduce barriers for the contributors (Gousios et al. 2015).

[3]https://opensource.guide/
[4]http://opensourcesurvey.org/2017/
[5]http://goo.gl/mZtz9X
[6]http://dx.doi.org/10.5281/zenodo.46063, Feb. 2016/

Through examining the type and volume of questions adopted in similar related research, the survey instrument produced for this research can leverage the accumulated efforts of other researchers in this space, thereby bringing increased quality to this research's survey instrument. The final survey instrument consists of:

- 3 questions - only one option to be selected
- 1 question - selection of multiple options including 'Others'
- 31 questions - selection on 11-point Likert scale
- 1 question - textual input as a paragraph

The survey instrument designed using Google Forms can be viewed in appendix C.1.

5.5 Evaluating the Survey Instrument

Evaluation of the survey instrument refers to pre-testing or pilot study and is primarily formulated on the following baselines (Kitchenham and Pfleeger, 2008):

1. To assess that the questions come across as intended by the researcher.
2. To assess the likely response rate and the effectiveness of the follow-up procedures.
3. To evaluate the reliability and validity of the instrument.
4. To ensure that our data analysis techniques match our expected responses.

The evaluation of this survey instrument follows two steps, internal review, and a pilot study. At first, supervisors evaluated the survey instrument internally, Dr. Paul Clarke and Prof. Rory O'Connor. Then a pilot study was conducted by selecting a smaller subset of participants from the original sample size of the

top 1020 OSS projects. The pilot study follows the same process and procedures defined for the survey as a check that everything works as envisaged and that data can be collected effectively. Furthermore, the efficacy of the survey instrument is evaluated in the pilot study. Section 5.8.5 articulates the details relating to validity concerns for the survey instrument.

5.6 Obtaining Valid Data

Validity of data obtained depends on the establishment of a robust data collection technique, which in this study is through a survey instrument. The sample or subset of a population that is representative of larger population helps in attaining precise and reliable findings (provide useful answers) and corrections (Kitchenham and Pfleeger, 2008). It is instructive to consider the target population and sampling procedure from the perspective of analysing data leading to any meaningful conclusions, similar to (Kitchenham and Pfleeger, 2008):

- Will the analysis results address the study objectives?
- Can the target population answer our research questions?

The target population for this survey is from the OSS community. The details on the selection of the target population with sample size appear in section 5.8.1. The minimum number of responses required from this survey study is 120 considering that every survey is complete with valid responses and the survey participant has an authentic profile.

5.7 Analysing the Survey Data and Reporting

Typical analyses compare different populations of respondents; analyse associations and trends, or the consistency of scores (Ciolkowski et al., 2003). The data analysis for quantitative and qualitative data will be performed independent of each other. Responses obtained from open questions may utilise memoing, and thematic coding to aggregate the findings. The quantitative analysis will partition responses into more homogenous groups and compare different populations of respondents based on their profile, and also analyse associations accordingly (as well as attempting to identify response trends that are common across all participants).

Data analysis also inspects responses from the participants for completeness or incompleteness, and consistency. Furthermore, in case a survey response is incomplete, deciding to include it or exclude it from the analysis without introducing any systematic bias. The findings from both types of analysis are combined to understand and elaborate knowledge-relevant activities in OSS projects and to evaluate and improve the canonical model of PKR practices. The details of the data analysis performed are presented in chapter 6.

The reporting of the survey involves the evaluation of the compiled survey results and disseminating information to the concerned parties in academia, in OSS communities, and more broadly in the software engineering field. The details of evaluations consolidated from the analysis of data collected from the survey are presented in chapter 7.

5.8 Conducting the Survey

This section elaborates the deployment of the survey for the data collection. The survey instrument was designed in section 5.3.4. The preceding step to the survey deployment is the selection of the participants in OSS projects. The following sections discuss the details on the selection of survey participants, and conducting the survey.

5.8.1 Contributor Selection from GitHub

GitHub is the world's largest collection of OSS (Borges et al., 2016), with 85 million repositories hosted and supporting a community of 28 million people to learn, to build and to share software, as reported by GitHub website on June 2018. The users of OSS projects on GitHub use stars to indicate their satisfaction, interest and appreciation for the project, which serves as a proxy measure of their popularity and the number of stars are observed to be directly related with number of forks, programming languages, users or organisations and domains (Borges et al., 2016). The process to extract contributors is summarised in Figure 5.2.

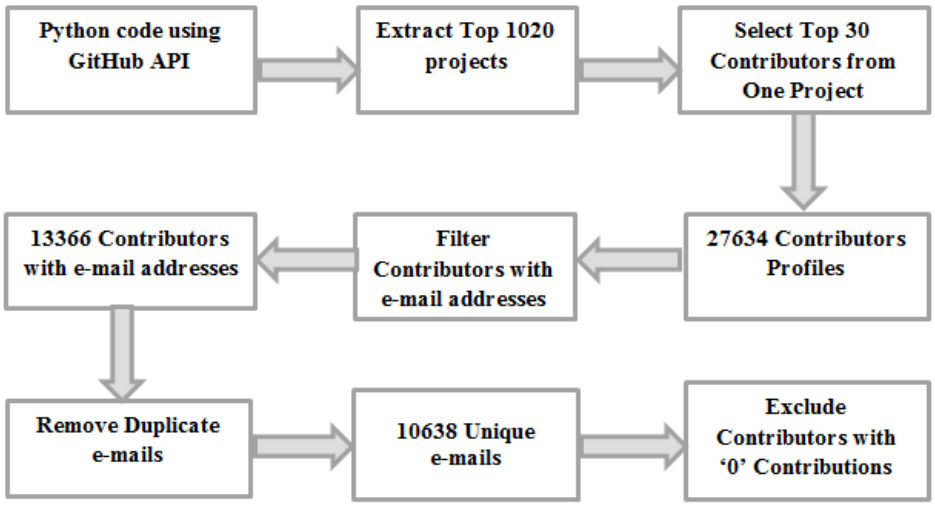

Figure 5.2: Process to extract survey participants from GitHub

The GitHub Search API functions similar to Google, while performing search

to find a specific item such as a user or a specific file in a repository. The design of the Google API attend to user's need and aim to find the specific results that best meets user's needs. The GitHub Search API provides up to 1,000 results for each search.

- The default limit when using the GitHub API to extract project repositories is 30x34=1020. For this work, the default limit is used to find the contributors to target.

Contributors were extracted with search criteria based on ranking of 'most stars' and by running a python script to correlate the e-mail addresses of contributors. The GitHub search then populates the top 1020 OSS projects [7], based on the criteria of number of stars, as a proxy measure for the popularity of projects, as shown on GitHub website portrayed in Figure 5.3.

- `https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories`

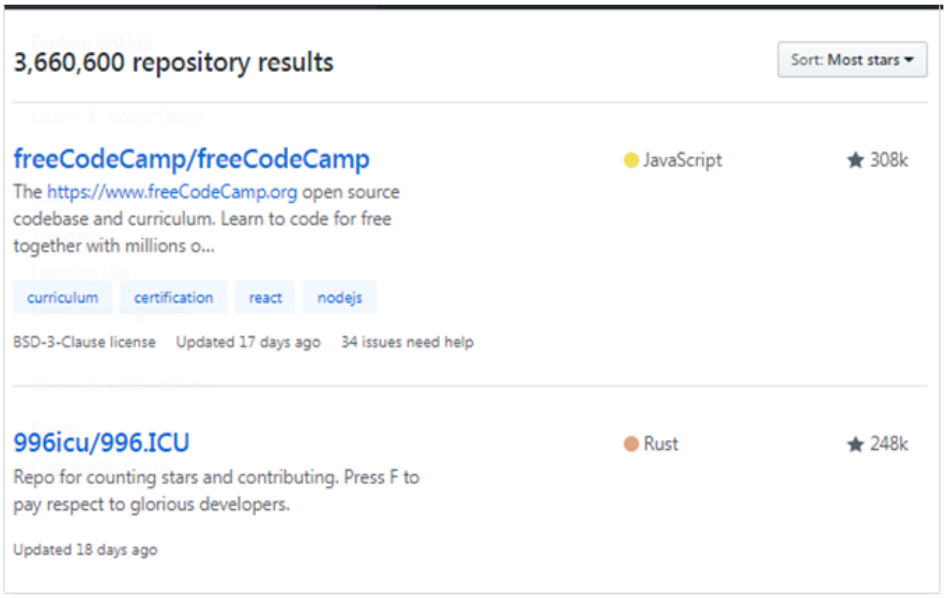

Figure 5.3: GitHub web page populated with projects with most stars

[7]Note: The data extraction was on 20-10-2018; the names of the projects that appear on the GitHub website will vary over time.

The GitHub Search API employed to search repositories are detailed on the following link:

- `https://developer.github.com/v3/search/#search-repositories`

Using GitHub API, the endpoint was queried to get all 34 pages, each page displays listings of 30 repositories by default. There are 34 pages extracted using GitHub API repositories. Each page has 30 repositories:

34 (pages) x 30 (repositories on each page) = 1020 repositories

Overall, 1020 repositories were extracted, out of 2,819,554[8] using the link:

- `https://github.com/search?o=desc&p=2&q=stars%3A%3E1&s=stars&type=Repositories`

There is a limit on the number of anonymous calls that can be made to GitHub. To overcome the issue of requests limits, an authentication personal access token was generated from the settings of a valid GitHub account. The generated personal access token was passed as a request parameter to increase the limit of anonymous API calls to 5000 per hour.

Each of 1020 project repositories were iterated by making a call to `contributors_url` to retrieve the details of contributors. Only the top 30 contributors in the each repository were selected, based on the number of contributions. Within the top 1020 projects, contributors from just a single project were not retrieved, because the project in question seemed to have an infinite number of contributors. Some projects has less than 30 contributors. If the number of contributors was less than 30 all contributors on the project were retrieved. Next, the details on the profiles of contributors were retrieved by iterating through `contributors_url` for every contributor. The profile details included contributors e-mail addresses and other relevant information. A total

[8]https://github.com/about

of 27634 contributors were extracted among whom 13366 have public e-mails addresses.

The results were displayed in JSON (JavaScript Object Notation), a readable format for structuring data. In order to convert JSON to csv, the Jupyter Notebook App was utilised to run python code. The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. A notebook kernel is a "computational engine" that executes the code contained in a Notebook document.

Also, the Jupyter Notebook App embeds the Pandas library, a software library written for the Python programming language for data manipulation and analysis. Notebook documents generate human-readable documents containing the analysis description and the results, for example in tables, as well as executable documents which can be run to perform data analysis. Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. The Pandas library object "dataframe" is used to store the JSON data structure and convert it to csv. The extracted names of the repositories with details on other attributes were verified from GitHub webpage:

- `https://github.com/search?o=desc&q=stars%3A%3E1&s=stars&type=Repositories`

5.8.2 Selection of Projects

The `public_repos` attribute extracted among contributor's profile indicates the number of contributions that contributor made to different repositories on GitHub. This is used as the criterion to select contributors considered to be experts in OSS contributing. From 13367 contributors who have valid e-mail addresses only those were selected who have contributed to repositories in the range of 1 to 3955. The highest number of contributions from a contributor to

GitHub repositories are 3955. Figure 5.4 gives an example of details on contributor's profile retrieved from GitHub.

email	login	name	prof_url
xxxxxxxx@gmail.com	xxxxx	Name 1	https://api.github.com/users/name1
xxxxxxxx@gmail.com	xxxxx	Name 2	https://api.github.com/users/name2
xxxxxxxx@gmail.com	xxxxx	Name 3	https://api.github.com/users/name3
xxxxxxxx@gmail.com	xxxxx	Name 4	https://api.github.com/users/name4
xxxxxxxx@gmail.com	xxxxx	Name 5	https://api.github.com/users/name5
xxxxxxxx@gmail.com	xxxxx	Name 6	https://api.github.com/users/name6
xxxxxxxx@gmail.com	xxxxx	Name 7	https://api.github.com/users/name7

proj_name	public_repos	repos_url
Project1/restful-authentic	168	https://api.github.com/users/Name 1/repos
Project1/restful-authentic	70	https://api.github.com/users/Name 2/repos
Project1/restful-authentic	34	https://api.github.com/users/Name 3/repos
Project1/restful-authentic	29	https://api.github.com/users/Name 4/repos
Project1/restful-authentic	23	https://api.github.com/users/Name 5/repos
Project1/restful-authentic	51	https://api.github.com/users/Name 6/repos
Project1/restful-authentic	27	https://api.github.com/users/Name 6/repos

Figure 5.4: Data relevant to contributors profiles on GitHub

It is observed that with an increasing number of contributions, the number of contributors tend to decrease. For example, 445 contributors made contributions ranging from 151 to 200 repositories. On the contrary 68 contributors made contributions ranging from 300 to 350 repositories.

5.8.3 Plain Language Statement (PLS)

In order to comply with General Data Protection Regulation (GDPR), a Plain Language Statement (PLS), was included in the beginning of the survey, where the participant has to give consent before taking the survey. The data collected from the survey is anonymous, i.e. the collected responses can not be traced back to the participants. In order to keep the survey anonymous, option for follow-ups on Google Form, which requires enabling the "collection of e-mail option", is disabled. The PLS statement and consent form can be viewed in the survey instrument document in Appendix C.1.

5.8.4 Sending Surveys using GMass

GMass is a marketing extension, used to mass e-mail campaigns that integrates with Gmail accounts. GMass suits the requirements of this work for issuing the survey, scheduling follow-ups as reminders and ensuring that they do not end up in the Spam folder of recipients. These measures effectively improve the response rate of the survey.

5.8.5 Phase - I: Pilot Survey

The pilot consisting of 50 surveys was conducted selecting contributors from 989 unique e-mail addresses with the selection criteria of 151 to 159 contributions in GitHub using GMass. The selection criteria for contributors ensured that they have enough experience contributing to OSS projects to attempt the survey and share their feedback. One e-mail address was not found to be valid. GMass (free subscription) limits sending more than 50 surveys in 24 hours. The follow-ups on the survey were set to one reminder after 3-days. A pilot survey response rate of 8% was achieved, which is considered positive given that no relationship existed between the researcher and the pilot participants. Furthermore, the pilot participants did not recommend changes to the survey but were provided with the opportunity to do so if desired. It should however be kept in mind that response rates while although informative, are alone not good proxies for study validity (Curtin et al., 2000; Keeter et al., 2006). Furthermore, a response rate not to be effectively considered a sufficient evidence to judge study quality and/or validity (Morton et al., 2012).

The pilot survey was successful and responses were collected in Google Forms. The setup for survey including GMass and Google Form was working in entirety. The next step was to send the survey to all contributors extracted from GitHub in Figure 5.2 by overcoming the sending limit on GMass. To extend the survey sending limit on GMass, a subscription was purchased from GMass, which enabled

the researcher to send 1950 e-mails every 24 hours.

5.8.6 Phase - II: Survey

The next stage after conducting the pilot survey was to issue the survey to the remaining contributors. During one week, a total of 10588 surveys were issued. The range of contributions represented by the `public_repos` field for the data collected from GitHub for contributors, was set to 1-3955 and excluding the e-mails already used to conduct pilot survey. The surveys were sent by placing all selected e-mail addresses in the 'To' field using Gmail compose e-mail option merged with GMass. Incomplete e-mail addresses were removed after attempting to locate correct e-mail address on GitHub profile.

- A total of 233 delivery failures and 158 bounces were recorded in the "bounce" folder of GMass.
- A total of 132 responses were received for the survey over the period of: 08-04-2019 to 15-06-2019
- The effective response rate of survey is: 132 / (10588-391) = 1.29%

Where 10588 is the number of OSS contributors with an e-mail address and 391 are the bounces. A selection from the collected data is depicted in Figure 5.5. The complete data collection is divided into four appendices: data on contributor profile and practices 1 - 4 appear in C.2 , data on practices 5 - 13 appear in C.3, data on practices 14 - 22 appear C.4, and data on practices 23 - 32 appear C.5.

Profile Questions 1-4					Questions on PKR Practices 1-4			
Please tick this box to indicate you consent to your input to this survey being used anonymously and you understand that your involvement is voluntary.	1. In total how many OSS projects have you worked on?	2. Which of the following activities apply to your OSS work?	3. How many years have you been contributing to OSS projects?	4. How much computer programming experience do you have?	1. Encourage pair programming and shared code ownership	2. Promote a policy that encourages peer review contributions from all project roles (irrespective of their role or seniority)	3. Use bug labeling so that contributors can effectively select tasks and make contributions (e.g. "suited for newcomers", "Feature xyz")	4. Ensure the presence of testing artefacts (e.g. unit tests, test scripts, test cases).
I consent	10+	Bug reporter,	10+ Years	10+ Years	8	7	10	8
I consent	10+	Bug reporter,	10+ Years	10+ Years	6	10	9	10
I consent	10+	Bug reporter,	10+ Years	10+ Years	3	9	7	9
I consent	10+	Bug reporter,	5 to 10 Year	10+ Years	5	7	7	9
I consent	10+	Bug reporter,	5 to 10 Year	5 to 10 Years	10	10	10	10

Figure 5.5: Data collected from the survey

Queries from contributors - There were e-mail queries from OSS contributors on the research along with an expression of interest to be informed on the outcome of this work. The overall assessment of the e-mail correspondence did not identify new items relevant for this research. A list is maintained by the researcher for OSS contributors to be informed on the outcome of this work and disseminate the findings.

5.9 Chapter Summary

This chapter discussed the process to design the survey instrument and the deployment of the survey instrument resulting in the data collection. The survey design process is based on seven steps, setting the objectives, survey

design, developing the survey instrument, evaluating the survey instrument, obtaining valid data, analysing the data, and survey reporting. The responses are based on open questions and close questions. The target OSS population is selected based on the popularity of OSS projects from GitHub. The selection of the participants from GitHub was by executing a sequence of programs to obtain the profiles of OSS contributors. The profiles were also verified for their accuracy from GitHub website by random selection of contributors.

The noted response rate for the survey is 1.29%. The main goal of the survey instrument is to evaluate and improve the PKR canonical model based on the OSS contributor feedback. The OSS contributor profiling will allow for an in-depth analysis of collected data and effectiveness of PKR practices in OSS projects. The following chapter 6 discusses data analysis performed on the collected data.

Chapter 6

Data Analysis

In this chapter, the data collected through survey instrument is analysed. This analysis commences by providing an overview of the collected data.

6.1 Data Analysis Overview

Of the 132 responses obtained, a total of 126 were deemed valid for analysis. The remaining six were disregarded from the study as they presented as problematic. Three of the discounted responses contained negative or disparaging commentary suggesting that the responses from these participants were not earnest in addressing the objectives of the study. In the case of a further three responses, the profile of the responses, either all fully agreeing or all fully disagreeing with the value of practices indicated that the respondents did not give due consideration to the practices, which was further confirmed by analysis of the timestamps for responses which were entered very rapidly by the participants. The remaining 126 responses were progressed to the detailed analysis stage.

6.1.1 Overview of Survey Participants

Certain participant profiling data was collected to enable a thorough analysis of the responses in the context of the experience and roles of the respondents in OSS. In terms of experience based on the number of OSS projects, 73% of the respondents report contributions to 5 or more different OSS projects can be

considered positive in terms of the informed view of survey participants. Refer to Table 6.1 for a breakdown on contributors' percentage of participation by number of OSS projects.

Table 6.1: Representation of OSS contributors profile in the survey population

Contributor's Profile	**Percentage**			
No. of Projects in OSS	1-5	5-10	10+	
	27%	**18%**	**55%**	
No. of Years in OSS	1-5	5-10	10+	
	43%	**28%**	**29%**	
No. of Years in Programming	1-2	2-5	5-10	10+
	2%	**17%**	**27%**	**54%**
Code Contributor	**19.2%**			
Tester	**8.17%**			
Bug Reporter	**16.6%**			
Maintainer	**14.19%**			
Committer	**14.19%**			
Reviewer	**12.3%**			
Document Writer/ Editor	**10.18%**			
Integrator	**4.47%**			
Others	**0.46%**			

In terms of OSS role types, some respondents reported contributing only in one role while other respondents reported contributing in two or more roles. Overall, 19.2% of the respondents contribute as code contributors, 8.1% contribute as testers, 16.6% contribute as bug reporters, 14.19% contribute as maintainers, 14.19% contribute as committers, 12.3% contribute as reviewers, 10.18% report contribute as document writers/ editor, 4.47% contribute as integrators, and 0.46% (3 responses) contribute as others. Refer to Table 6.1 for a breakdown of reported role types. A broad spectrum of role types are represented in the participant population is considered positive to the research effort, as there is no single viewpoint dominating the population.

In this research, the term most experienced in OSS relates to contributors who have selected 10+ on any of the three attributes, e.g. contributions on 10+ projects, have been contributing for 10+ years or have programming experience of 10+ years. Alternatively, the least experienced group of contributors are those with the selection of 1-5 for the first two attributes and 1-2 for the third attribute.

The remainder of the selection including, e.g. 2-5, 5-10, formulate the group of contributors with middle level experience.

In terms of experience based on number of years in OSS projects, 57% reported to have contributions in OSS projects for more than 5 years. In terms of programming experience, 2% contributors report 1-2 years, and 72% report more than 5 years. Table 6.1 depicts the breakdown of percentage on contributors' participation by the number of years in computer programming.

As a general comment, the profiling demonstrates that a broad spectrum of OSS contributors have participated in this study. The data collected is representative of various OSS contributors contributing in different roles, with varying levels of experience in programming ranging from relatively inexperienced to highly experienced. The dynamic profiles of respondents will allow for a detailed comparison of the views' of different types of OSS contributors, while also bringing balance to the research as a whole, wherein no single perspective is over represented in the research population.

Of the 126 participants under analysis, the following profiles were observed:

- The majority of the contributors are experienced in OSS projects, about 55% have worked on more than 10 projects.
- The majority of contributors contributing to OSS projects are experienced above 4 years, about 57% population is inclusive of contributors having an experience of 5-10 and 10+ years in OSS projects.
- Regarding programming experience, 54% of contributors have more than 10 years of experience and 27% have 5 to 10 years of experience.
- In the role type, there is a general balance of OSS activities in the respondents with no one role type dominating the sample, e.g. code contributors 19.2%, testers 8.17%, bug reporters 16.6%, maintainers 14.19%, committers 14.19%, reviewers 12.34%, document writers/ Editors 10.18%, and integrators 4.47%.

6.1.2 Likert-Type Scale

Likert developed a procedure for measuring attitudinal scales to address the difficulty of measuring character and personality traits referred to as a Likert scale (Likert, 1932). The original Likert scale was constituted on five response alternatives: strongly approve (1), approve (2), undecided (3), disapprove (4), and strongly disapprove (5) and during data analysis a composite score was calculated from a set of questions representing attitudinal scale (Boone and Boone, 2012). The evaluation of practices obtained in this work is from survey based on Likert-type items, and during data analysis, the researcher does not intend to calculate a composite score for responses obtained against a set of practices, which is applicable for the analysis of responses gathered for a Likert scale (Boone and Boone, 2012). As argued that a single Likert-type item inquires of the respondent about their choice among several ordered alternatives presented and each Likert-type item reflects a discrete approximation of the continuous latent or hidden variable (Clason and Dormody, 1994).

The emphasis here is that the discrete nature of the response is acknowledged while performing an analysis of single items from Likert scales and implication of ignorance of the discrete nature of responses can produce inferential errors (Clason and Dormody, 1994). Accordingly, in Likert-type items, "number(s) are assigned in order to express a "greater than" relationship; however, how much greater is not implied and because of these conditions, Likert-type items fall into the ordinal measurement scale" (Boone and Boone, 2012). Furthermore, the recommended data analysis procedures for ordinal measurement scale items include a mode or median for central tendency and frequencies for variability (Boone and Boone, 2012). The technique to rank practices based on the evaluations of respondents is elaborated in section 6.2.

As per details available in earlier chapters, the evaluation of the 31 practices is conducted using a Likert-type items scale. The values on the scale range from

0 to 10 (0 indicates that a practice is not effective practice in OSS projects, and 10 indicates that a practice is highly effective in OSS projects). A Likert-type items scale is shown in Figure 6.1.

Figure 6.1: 11-point Likert Scale used for conducting OSS community survey

6.1.3 Data Summary

The data is collected as responses on Likert-type items against four profile questions: the contributor's contribution to the number of OSS projects, the activities in OSS projects, the number of years contributing to OSS projects, and the number of years in computer programming. The data collected from survey indicates that most of the contributors perform more than one activity in OSS projects.

The contributor activities in OSS projects are categorised as bug reporters (108), code contributors (125), committers (92), maintainers (92), reviewers (80), document writers, or editors (66), testers (53), integrators (29), and others (3). The data collection from the survey represents contributors from the major roles in OSS projects. Contributors who identify themselves as others were three in number, two of which are community managers and one relates to identification of new ideas in OSS projects. The categorisation by the number of years in OSS projects indicate that majority of contributors have contributed for more than 10 years. Similarly, categorisation by the number of years in OSS projects shows that the majority of contributors have been active for 1 to 5 years. Categorisation by the number of years in computer programming shows that the majority of contributors have more than 10 years of programming experience. The responses obtained against four profile questions are summarised in Table 6.2.

Table 6.2: Overview of the survey data obtained

Variable	Value	Frequency
Number of OSS Projects	1-5 years	34 (27 %)
	5-10 Years	23 (18.2%)
	10+ Years	69 (54.8%)
Role types	Bug reporter	108 (85.7%)
	Code contributor	125 (99.2%)
	Maintainer	92 (73%)
	Reviewer	80 (63.5%)
	Committer	92 (73%)
	Document Writer and Editor	66 (52.4%)
	Tester	53 (42.1%)
	Integrators	29 (23%)
	Others	3 (2.4%)
Number of years in OSS Projects	1-5 years	54 (42.9%)
	5-10 Years	35 (27.8%)
	10+ Years	37 (29.4%)
Number of years in Computer Programming	1-2 years	2 (1.6%)
	2-5 Years	22 (17.5%)
	5-10 Years	34 (27%)
	10+ years	68 (54%)

Figures 6.2, 6.3, and 6.4 depicts the distribution of responses obtained for practices along with their counts using Likert-type item scale. As a general observation, the data collected for each practice as evident from plots in the Figures 6.2, , 6.3, and 6.4, is not normally distributed. Normal distribution is bell-shaped, and perfectly symmetrical around its centre (Petrie and Sabin, 2019). At the first glimpse of data, the distribution trend for most of the practices presents with left skewness, for example Practice 2 (P2), Practice 3 (P3), Practice 4 (P4) and others. A trend of distribution around the middle value is also noticed for the responses obtained on the Likert-type scale for Practice 6 (P6), Practice 16 (P16), Practice 17 (P17), Practice 23 (P23), Practice 26 (P26), Practice 29 (P29),

and Practice 30 (P30). The descriptions of practices against abbreviation P1, P2, P3 and so on, appear in Appendix D.1.

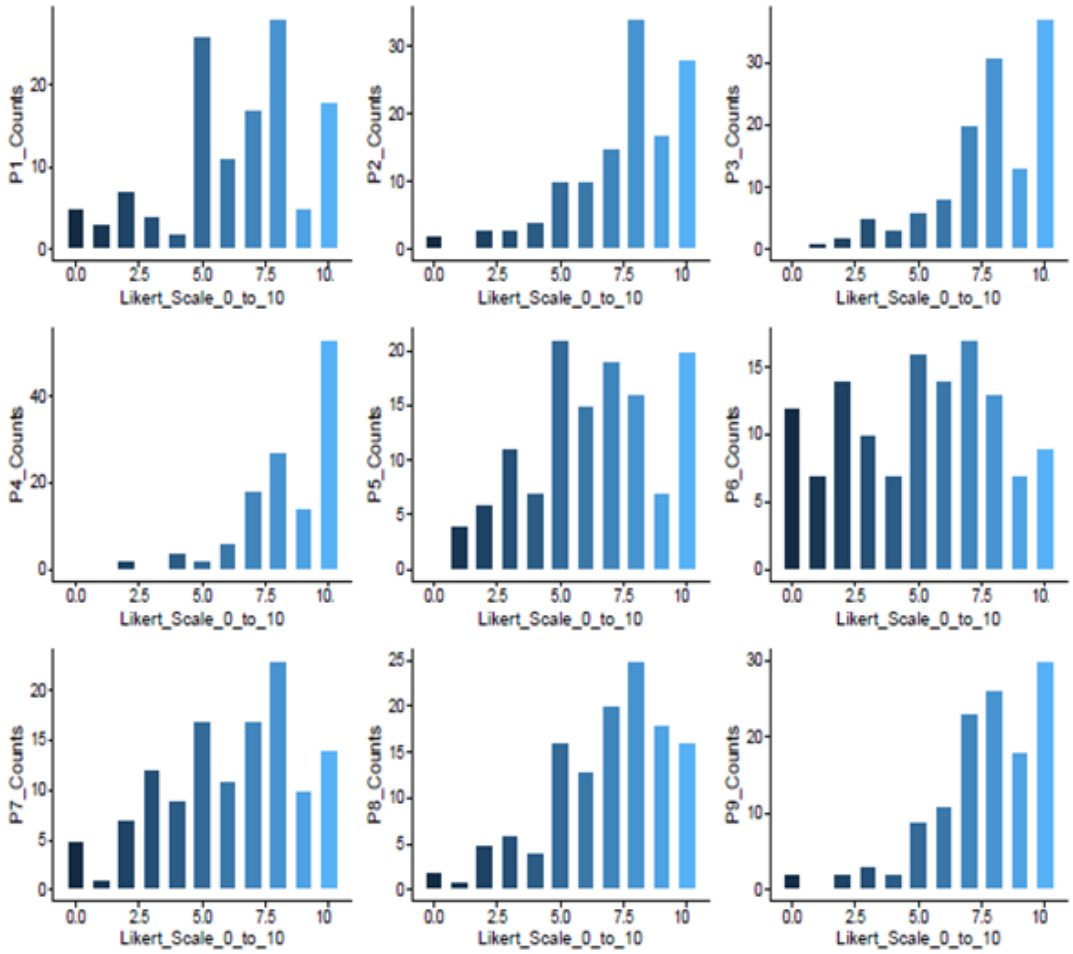

Figure 6.2: The distribution of responses for practices (1-9) and frequency on the Likert scale (0-10)

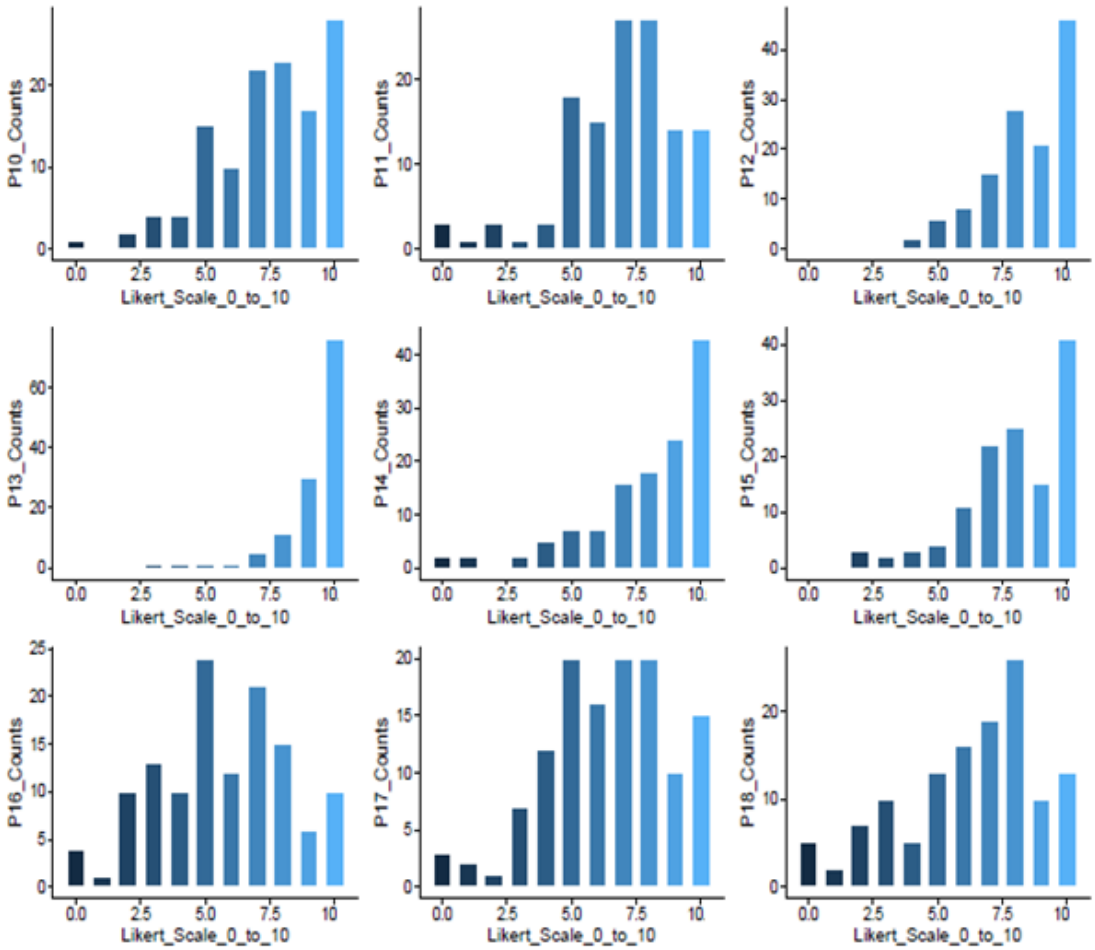

Figure 6.3: The distribution of responses for practices (10-18) and frequency on Likert scale (0-10)

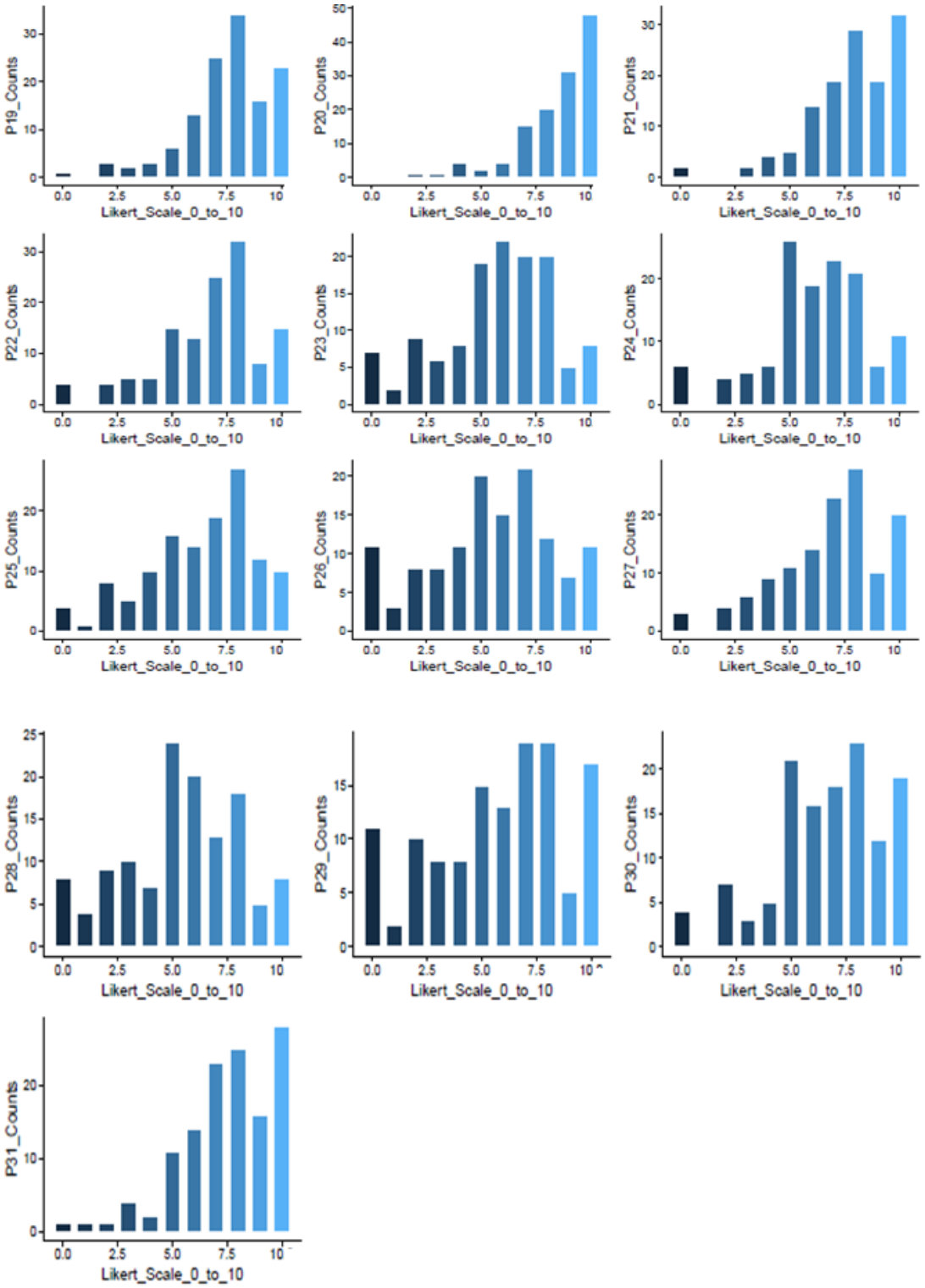

Figure 6.4: The distribution of responses on practices (19-31) and frequency on Likert scale (0-10)

6.2 Ranking Technique

In this section, the view of data is presented in accordance with median, mean, standard deviation, and average deviation. Median gives the central location of the data. Varying the highest or the lowest value in data will not make any difference to the value of median. The value of median remains the same even with added outliers and skewed data (Petrie and Sabin, 2019). When data distribution is not symmetrical or open-ended, median is the right measure of central tendency (Sundaram et al., 2010). However, calculation of mean includes all data values or all information in the data and so it is thought to be more efficient than the median, which does not include all values in the data (Petrie and Sabin, 2019).

Standard Deviation (SD) represents the spread or variability of data values by calculating the average value of the squares of the distances between the data values and mean (Ross, 2014). Another statistical measure of dispersion is Mean Absolute Deviation (MAD), which finds out the absolute distance between each data point and the mean and provides the measure of expected deviation of a random variable from its mean value (Yager and Alajlan, 2014). In this analysis, the values of mean, median, SD, and MAD are calculated for each practice and ranked. The values of mean and median are ranked in descending order and values of SD and MAD are ranked in ascending order. All four ranking measures are utilised to perform a final ranking of practices into a hierarchy.

6.2.1 Designing Practice Ranking Scheme

The data collected as responses to evaluate 31 practices is plotted to observe the trend of evaluation in each practice. In the Figure 6.5, the median bar chart illustrates these trends; x-axes represent the practice number and y-axes represent the values of calculated measures (i.e. median, mean, SD and MAD). The medians for most of the practices, with the exception of Practice 6, are around or above 6.

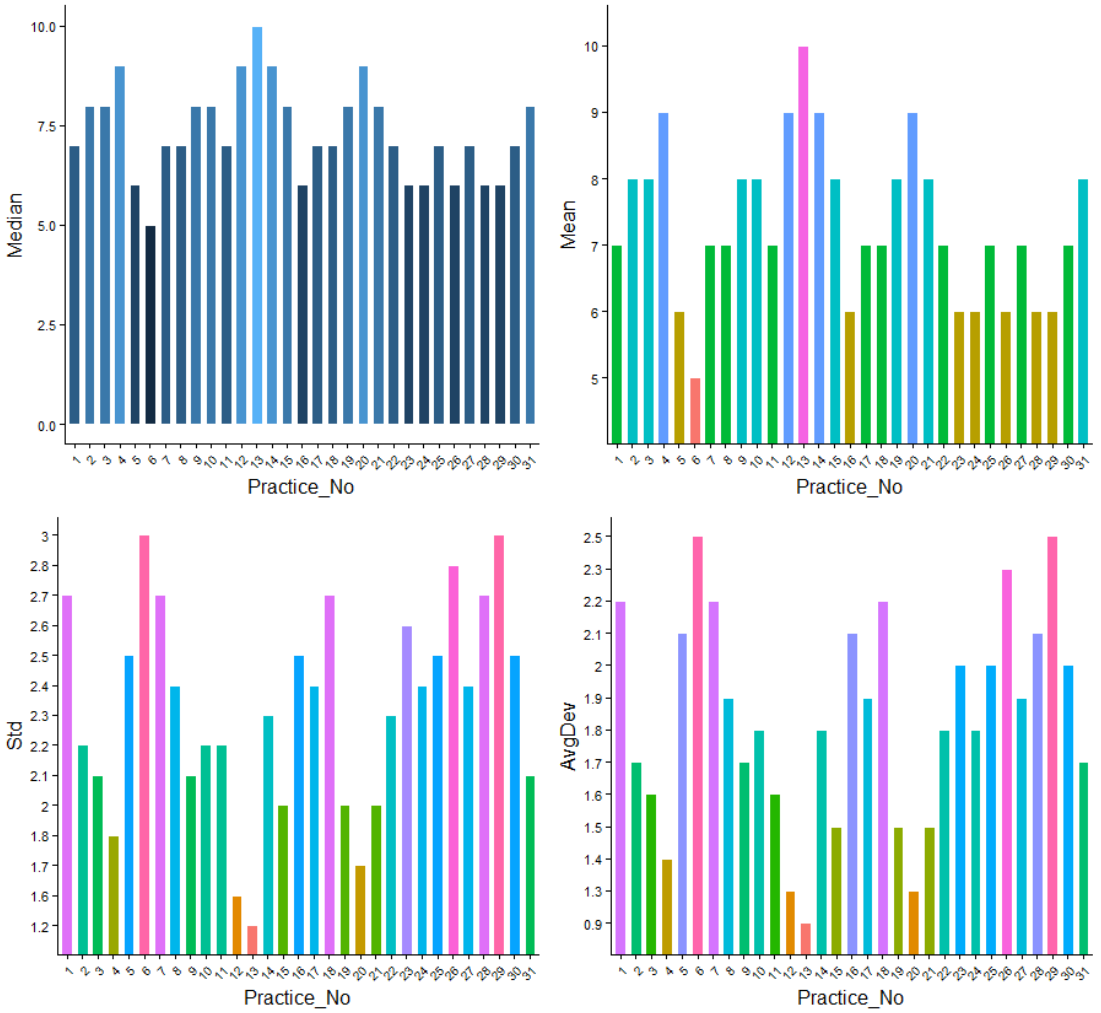

Figure 6.5: Distribution of responses obtained for 31 practices based on Median, Mean, Std (SD), AvgDev (MAD)

Incorporating the calculations from mean to analyse the evaluation trend of practices does not show a significant change from the median representation, rather in general, a minor decrease in values of mean but almost equal to median.

Comparison between mean and median generally reveals information about the shape of the distribution. "A distribution, or data set, is symmetric if it looks the same to the left and right of the center point" (Ramachandran and Tsokos, 2009). Inferring on the distribution of data obtained from OSS contributors while evaluating 31 practices, and considering median as a center point, if mean equals median, then it is a symmetric distribution, if mean is greater than median, it is a positively skewed distribution and if mean is less than median, it is negatively skewed. In the data set obtained as responses of

OSS contributors while evaluating 31 practices, distribution for most of the practices is negatively skewed.

Figure 6.5 also includes the calculated values of SD for all practices. The standard deviation is a measure of statistical dispersion from the centre of data around the mean (Ross, 2014). In order to rank practices for their effectiveness an understanding is required about the dispersion of opinions among respondents for each practice around mean. A preferable practice that is regarded a high rating based on the evaluation of respondents is closer to the mean and will show low variance from the mean as indicated by SD value. Alternatively, MAD (AvgDev) calculates the average distance from a set of data between each data value and the mean. The bar plots of mean, median, SD and MAD are depicted in Figure 6.5 .

Table 6.3 presents the basic ranking of practices. In order to obtain a single rank against each row, a comparison on the ranking obtained for each practice based on mean, median, SD, and MAD is performed. To arrange the practices into a ranked order, the mean and median values are considered in combination, with the practices first being sorted based on their mean values. Subsequently, the median value is considered, and any instances where there appears to be a large deviation from the mean to the median are examined for specific reasons for this deviation. This involves also looking to the SD and MAD values to get a sense for the observable variation in the generated mean values. Using these additional perspectives, it is possible to refine the practice ranking, especially in cases where the presented mean values are equal or almost equal. This process is used in the various rankings applied in this thesis, and the outcome of this process as alied to the survey population as a whole is presented in Table 6.3.

Although various approaches to grouping the practices were explored, when the various categorisations were considered (the results of such analysis are presented later in this chapter) there was no one formula that worked consistently well for all categorisations and therefore, it was decided to simply

group the data based on the absolute number. Accordingly, practices with a mean value of between 5 and 6 were grouped together, while practices between 6 and 7 were also grouped together. This, it is accepted, is a relatively crude grouping strategy but it was the most effective one identified as being useful across all categorisations and allows for a single interpretation across all categorisations which aids in consistency of understanding.

Table 6.3: Ranking of overall practices

Rank No.	Calculations				Practice No.
	Mean	Median	SD	MAD	
1	9.294	10	1.207	0.852	P13
2	8.564	9	1.69	1.309	P20
3	8.444	9	1.583	1.321	P12
4	8.444	9	1.782	1.432	P4
5	8.008	8	1.974	1.533	P15
6	7.992	9	2.296	1.757	P14
7	7.833	8	2.011	1.529	P21
8	7.818	8	2.111	1.616	P3
9	7.643	8	2.148	1.658	P9
10	7.564	8	1.97	1.49	P19
11	7.548	8	2.233	1.726	P2
12	7.516	8	2.112	1.673	P31
13	7.444	8	2.167	1.758	P10
14	6.929	7	2.184	1.633	P11
15	6.865	7	2.378	1.899	P8
16	6.825	7	2.407	1.901	P27
17	6.738	7	2.34	1.809	P22
18	6.635	7	2.51	2.025	P30
19	6.421	7	2.388	1.947	P17
20	6.357	7	2.658	2.154	P1
21	6.318	6	2.526	2.1	P5
22	6.294	7	2.501	2.048	P25
23	6.254	7	2.65	2.154	P18
24	6.159	6	2.375	1.833	P24
25	6.119	7	2.682	2.244	P7
26	5.738	6	2.999	2.49	P29
27	5.73	6	2.553	2.02	P23
28	5.659	6	2.515	2.077	P16
29	5.437	6	2.827	2.296	P26
30	5.397	6	2.663	2.137	P28
31	5.024	5	2.995	2.517	P6

Nevertheless, the limitations of this approach are discussed in the limitations section of this thesis. This grouping is also identified in Table 6.3, as indicated by the enclosed areas: e.g. rank 1-5 is in group 1, rank 6-13 is in group 2.

The groupings corresponding to Table 6.3 are depicted in Figure 6.6. The segregation of practices in is denoted by a line in Table 6.3. The practices with highest rankings are placed in the topmost layer (Layer 3) of the pyramid in Figure 6.6. The middle layer of pyramid constitutes cluster of practices highlighted in Layer 2 and Layer 1. The last layer, Layer 0 or the base of the pyramid contains practices that are least preferable by OSS contributors highlighted as darker grey. The description of practices against each practice number appears in Appendix D.1.

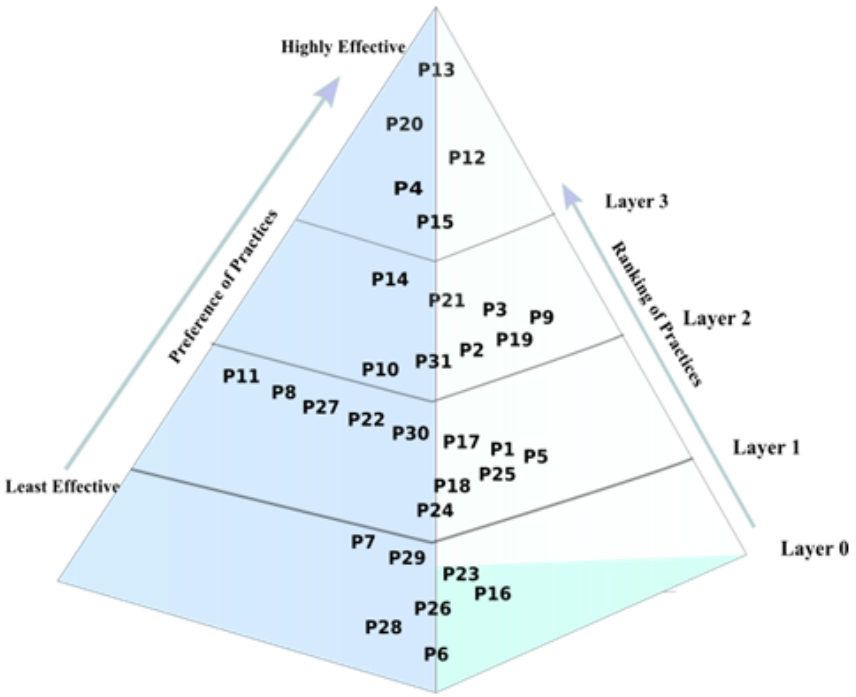

Figure 6.6: Ranking pyramid - the overall ranking of practices

6.3 Categorical Ranking of Practices

This section presents the ranking of practices based on contributor categories outlined in section 6.2 of this chapter. The ranking scheme presented in section

6.2 is applied throughout in order to group practices based on the effectiveness reported by contributors in different categories.

6.3.1 Ranking of Practices Based on the Number of OSS Projects

In Figure 6.7, mean, median, SD, and MAD values are illustrated, to demonstrate the effectiveness of practices by three groups of OSS contributors with varying experience based on the number of OSS projects namely: 1-5, 5-10, and 10+ projects.

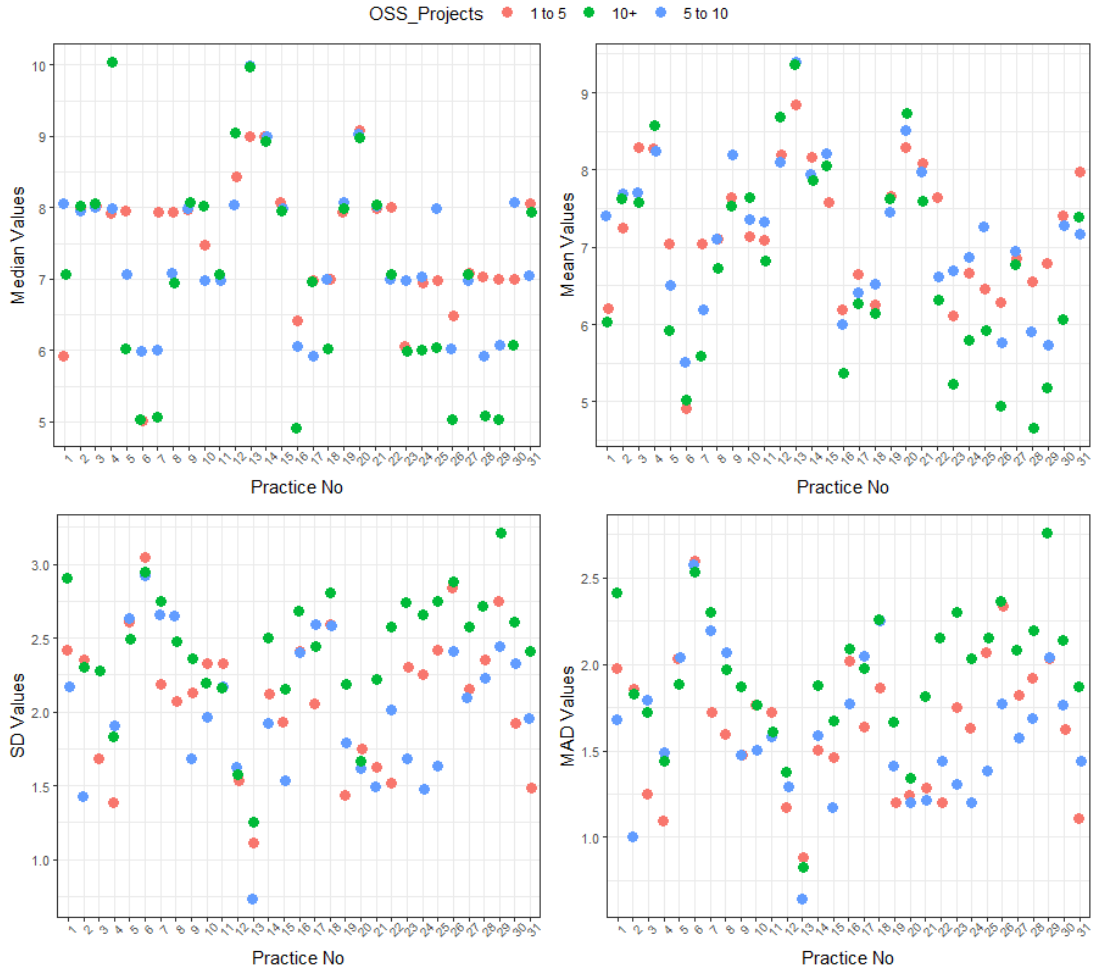

Figure 6.7: The disparity among three categories of contributors based on the number of OSS projects and values obtained for mean, median, SD, and MAD

.

Tables 6.4, 6.5, and 6.6 entail the categorical ranking of practices using mean,

median, SD, and MAD values, based on contributors' contributions to the number of projects. The contributors among all categories, 1-5, 5-10, and 10+, who base their experience on the number of OSS projects rank P13 at the highest place. The segregation similar to Table 6.3 is applied to rankings obtained in Table 6.4, 6.5, and 6.6 resulting into four levels, level 0, level 1, level 2 and level 3.

Table 6.4: Categorical ranking of practices based on 1-5 numbers of OSS projects

Practice Rank No.	Calculations 1-5 OSS Projects				Practice No.
	Mean	Median	SD	MAD	
1	8.912	9	1.19	0.836	P13
2	8.235	8	1.394	1.055	P4
3	8.235	9	1.742	1.28	P20
4	8.206	8	1.647	1.23	P3
5	8.176	8.5	1.585	1.235	P12
6	8.176	9	2.139	1.498	P14
7	8.088	8	1.658	1.28	P21
8	7.912	8	1.464	1.114	P31
9	7.735	8	1.582	1.253	P22
10	7.647	8	1.937	1.495	P15
11	7.647	8	2.087	1.457	P9
12	7.588	8	1.52	1.201	P19
13	7.412	7	1.877	1.554	P30
14	7.235	8	2.425	1.841	P2
15	7.118	8	2.129	1.633	P8
16	7.088	7.5	2.275	1.853	P10
17	7.088	8	2.621	2.083	P5
18	7.029	8	2.236	1.732	P7
19	7.029	7	2.303	1.678	P11
20	6.853	7	2.162	1.753	P27
21	6.824	7	2.702	2.1	P29
22	6.676	7	2.07	1.734	P17
23	6.618	7	2.243	1.699	P24
24	6.471	7	2.428	1.976	P28
25	6.412	7	2.488	2.045	P25
26	6.235	6.5	2.323	1.941	P16
27	6.235	7	2.547	1.927	P18
28	6.235	6.5	2.829	2.353	P26
29	6.206	6	2.496	1.924	P1
30	6.118	6	2.306	1.779	P23
31	4.853	5	3.056	2.585	P6

Table 6.5: Categorical ranking of practices based on 5-10 numbers of OSS projects

Practice Rank No.	Calculations 5-10 OSS Projects				Practice No.
	Mean	Median	SD	MAD	
1	9.435	10	0.788	0.639	P13
2	8.478	9	1.563	1.24	P20
4	8.217	8	1.536	1.183	P15
3	8.217	8	1.93	1.531	P4
5	8.174	8	1.642	1.327	P12
6	8.13	8	1.714	1.44	P9
7	8	8	1.537	1.304	P21
8	7.913	9	1.952	1.671	P14
9	7.783	8	2.354	1.743	P3
10	7.652	8	1.465	1.093	P2
12	7.435	7	1.879	1.584	P10
11	7.435	8	2.107	1.682	P1
13	7.391	8	1.777	1.331	P19
15	7.261	8	1.711	1.467	P25
14	7.261	7	2.115	1.599	P11
16	7.217	8	2.411	1.841	P30
17	7.13	7	1.914	1.463	P31
18	7.087	7	2.557	2.091	P8
19	7	7	2.132	1.652	P27
20	6.783	7	1.413	1.115	P24
22	6.609	7	1.751	1.365	P23
21	6.609	7	1.924	1.52	P22
23	6.478	7	2.556	1.981	P5
24	6.435	7	2.626	2.178	P18
25	6.348	6	2.534	1.958	P17
26	6.217	6	2.645	2.14	P7
28	5.913	6	2.234	1.664	P28
27	5.913	6	2.353	1.845	P16
29	5.783	6	2.449	1.83	P26
30	5.696	6	2.476	1.996	P29
31	5.478	6	2.952	2.544	P6

Table 6.6: Categorical ranking of practices based on 10+ numbers of OSS projects

Practice Rank No.	Calculations 10+ OSS Projects				Practice No.
	Mean	Median	SD	MAD	
1	9.435	10	1.3	0.836	P13
2	8.754	9	1.701	1.277	P20
3	8.667	9	1.55	1.314	P12
4	8.623	10	1.903	1.513	P4
5	8.116	8	2.118	1.712	P15
6	7.928	9	2.493	1.921	P14
7	7.667	8	2.356	1.865	P2
8	7.652	8	2.293	1.738	P21
9	7.638	8	2.229	1.764	P3
10	7.623	8	2.21	1.741	P10
11	7.609	8	2.231	1.676	P19
12	7.478	8	2.305	1.806	P9
13	7.449	8	2.416	1.94	P31
14	6.768	7	2.164	1.651	P11
15	6.754	7	2.626	2.068	P27
16	6.667	7	2.447	1.961	P8
17	6.319	7	2.506	2.024	P17
18	6.29	7	2.635	2.107	P22
19	6.203	6	2.742	2.235	P18
20	6.072	7	2.835	2.33	P1
21	6.058	6	2.689	2.151	P30
22	5.913	6	2.661	2.19	P25
23	5.884	6	2.404	1.947	P5
24	5.725	6	2.617	2.076	P24
25	5.638	5	2.807	2.318	P7
26	5.29	5	2.624	2.145	P16
27	5.246	6	2.799	2.294	P23
28	5.217	5	3.185	2.684	P29
29	4.957	5	3.007	2.488	P6
30	4.928	5	2.871	2.293	P26
31	4.696	5	2.719	2.159	P28

The pyramid in Figure 6.8 depicts the preference hierarchy of practices by respondents divided in three groups based on the number of OSS projects. The highest preference by OSS Contributors appear in the topmost layer of the pyramid, referred to as Layer 3. The bottom layer or the base of the pyramid contains the cluster of practices that are reported as least effective by OSS

contributors across the different categories. The description of practices against each practice number appears in Appendix D.1.

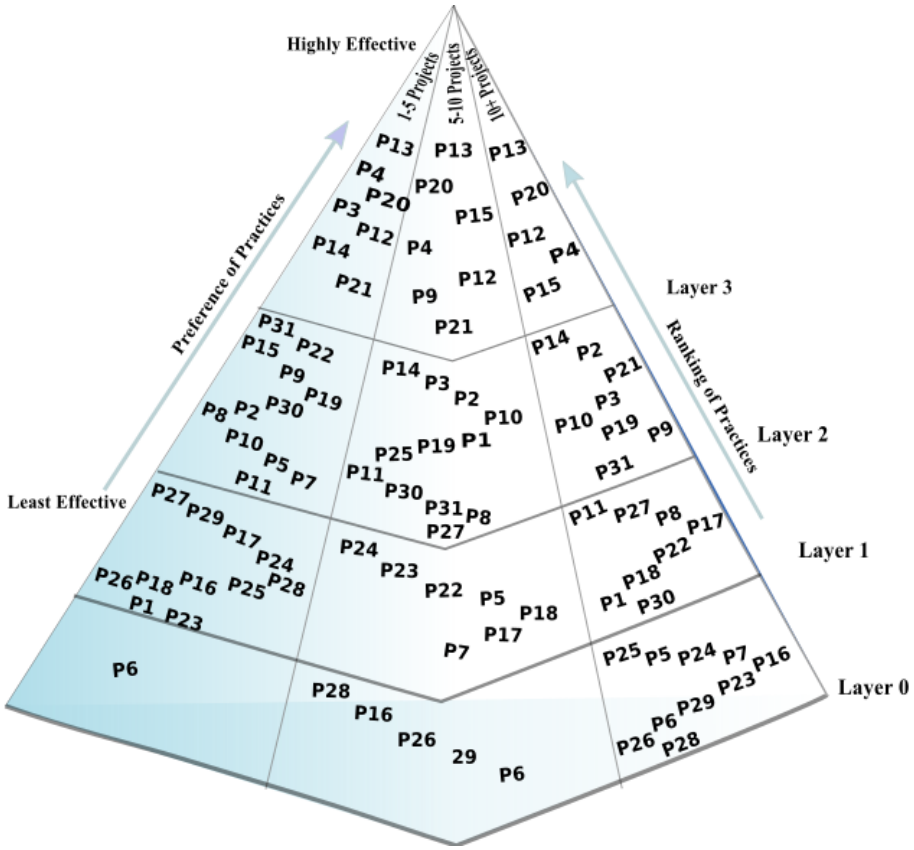

Figure 6.8: Hierarchy of practices based on the number of OSS projects

6.3.2 Ranking Practices Based on Number of Years in OSS

The opinions on the effectiveness of practices for contributors with varying number of years, 1-2, 5-10, 10+ in OSS projects are depicted in Figure 6.9 using mean, median, SD, and MAD values.

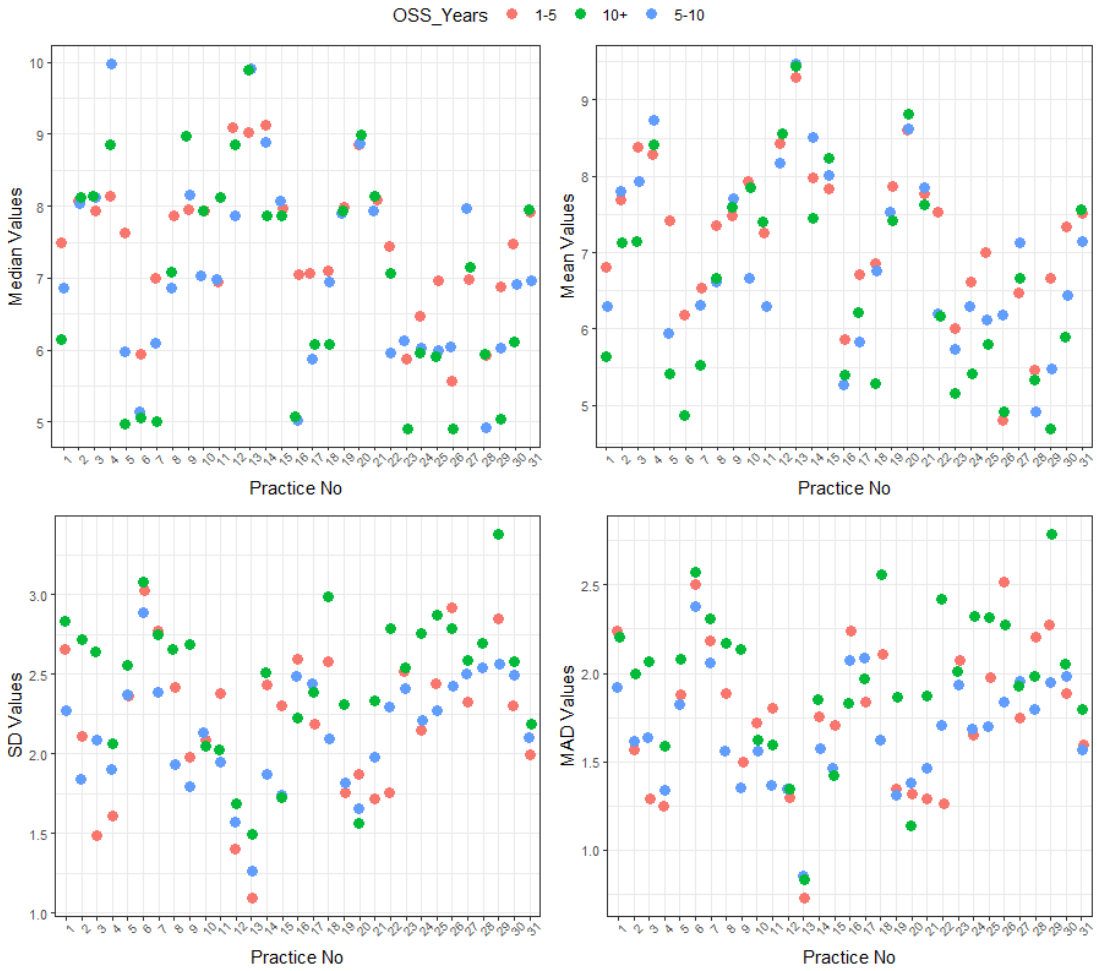

Figure 6.9: The disparity among three categories of contributors based on the number of years OSS and values obtained for mean, median, SD, and MAD

The ranking of practice effectiveness according to the population of contributors, categorised by the number of years (1-5, 5-10, and 10+) in OSS projects, is shown in Tables 6.7, 6.8, 6.9.

Table 6.7: Categorical ranking of practices based on 1-5 numbers of years in OSS

Practice Rank No.	Calculations 1-5 Years in OSS				Practice No.
	Mean	Median	SD	MAD	
1	9.27	9	1.005	0.802	P13
2	8.541	9	1.827	1.377	P20
3	8.459	9	1.488	1.218	P12
4	8.297	8	1.497	1.245	P3
5	8.243	8	1.543	1.189	P4
6	8.054	9	2.351	1.794	P14
7	7.919	8	2.212	1.716	P15
8	7.892	8	2.123	1.747	P10
9	7.811	8	1.725	1.355	P21
10	7.784	8	1.81	1.338	P19
11	7.649	8	2.141	1.573	P2
12	7.595	7.5	1.723	1.352	P22
13	7.568	8	2.031	1.499	P9
14	7.568	8	2.048	1.529	P31
15	7.351	7.5	2.324	1.926	P30
16	7.351	7.5	2.393	1.907	P5
17	7.297	8	2.448	1.895	P8
18	7.243	7	2.379	1.778	P11
19	6.919	7	2.367	1.932	P25
20	6.892	7.5	2.701	2.167	P1
21	6.838	7	2.595	2.091	P18
22	6.784	7	2.272	1.76	P17
23	6.622	6.5	2.097	1.63	P24
24	6.595	7	2.785	2.209	P29
25	6.541	7	2.261	1.747	P27
26	6.541	7	2.703	2.162	P7
27	6.135	6	3.046	2.539	P6
28	5.946	6	2.606	2.003	P23
29	5.892	7	2.621	2.157	P16
30	5.541	6	2.732	2.235	P28
31	4.838	5.5	2.973	2.444	P26

Table 6.7 represents the ranking of PKR practices based on 1-5 years of contributor's experience in OSS projects. The contributors with an experience of 1-5 years in OSS are considered relatively inexperienced to contributors with 10+ years in OSS. Considering the experience gained with number of years in OSS projects, the ranking PKR practices among contributors can differ from the rather experienced contributors due to the preference of certain practices over others. A detailed evaluation of PKR practices based on contributors experience is presented in chapter 7.

Table 6.8: Categorical ranking of practices based on 5-10 numbers of years in OSS

Practice Rank No.	Calculations 5-10 Years in OSS				Practice No.
	Mean	Median	SD	MAD	
1	9.4	10	1.193	0.789	**P13**
2	8.829	10	1.823	1.378	**P4**
3	8.571	9	1.685	1.339	**P20**
4	8.457	9	1.853	1.482	**P14**
5	8.143	8	1.63	1.355	**P12**
6	8.029	8	1.823	1.404	**P15**
7	7.886	8	2.153	1.693	**P3**
8	7.829	8	1.902	1.53	**P2**
9	7.829	8	2.007	1.463	**P21**
10	7.714	8	1.792	1.388	**P9**
11	7.486	8	1.853	1.388	**P19**
12	7.2	7	2.139	1.6	**P31**
13	7.086	8	2.548	2.021	**P27**
14	6.686	7	2.111	1.656	**P18**
15	6.629	7	2.157	1.589	**P10**
16	6.543	7	1.945	1.613	**P8**
17	6.486	7	2.513	1.902	**P30**
18	6.371	7	2.327	1.861	**P1**
19	6.343	6	2.425	2.144	**P7**
20	6.314	7	1.891	1.412	**P11**
21	6.229	6	2.327	1.778	**P22**
22	6.2	6	2.153	1.657	**P24**
23	6.2	6	2.471	1.863	**P26**
24	6.114	6	2.298	1.775	**P25**
25	5.914	6	2.306	1.86	**P5**
26	5.886	6	2.506	2.016	**P17**
27	5.771	6	2.486	1.944	**P23**
28	5.514	6	2.501	1.984	**P29**
29	5.257	5	2.559	1.993	**P16**
30	4.829	5	2.479	1.873	**P28**
31	4.8	5	2.795	2.377	**P6**

Table 6.9: Categorical ranking of practices based on 10+ numbers of years in OSS

Practice Ranking No.	Calculations 10+ Years in OSS				Practice No.
	Mean	Median	SD	MAD	
1	9.378	10	1.479	0.907	P13
2	8.811	9	1.488	1.164	P20
3	8.568	9	1.676	1.379	P12
4	8.459	9	2.036	1.616	P4
5	8.324	8	1.733	1.468	P15
6	7.811	8	2.093	1.695	P10
7	7.703	8	2.414	1.845	P21
8	7.595	9	2.63	2.136	P9
9	7.514	8	2.545	1.93	P14
10	7.486	8	2.194	1.854	P31
11	7.459	8	2.317	1.836	P19
12	7.405	8	2.061	1.572	P11
13	7.216	8	2.637	2.048	P2
14	7.081	8	2.629	2.124	P3
15	6.73	7	2.524	2.012	P27
16	6.73	7	2.642	2.105	P8
17	6.189	6	2.355	1.901	P17
18	6.162	7	2.853	2.356	P22
19	5.865	6	2.594	2.085	P30
20	5.838	6	2.824	2.337	P25
21	5.703	6	2.817	2.283	P1
22	5.486	6	2.825	2.339	P24
23	5.459	5	2.588	2.089	P5
24	5.459	5	2.834	2.335	P7
25	5.351	5	2.251	1.823	P16
26	5.351	6	3.011	2.558	P18
27	5.297	6	2.686	2.057	P28
28	5.243	5	2.532	1.98	P23
29	4.973	5	2.853	2.247	P26
30	4.811	5	3.143	2.603	P6
31	4.703	5	3.415	2.846	P29

The pyramid in Figure 6.10 depicts the hierarchy of the practices by the preference of three groups based on the number of years contributing to OSS projects. The group of practices with highest preference by OSS Contributors appear in the topmost layer of the pyramid in Figure 6.10. The practices that are least preferable by OSS contributors cluster at the base of the pyramid. The description of practices against each practice number appears in Appendix D.1.

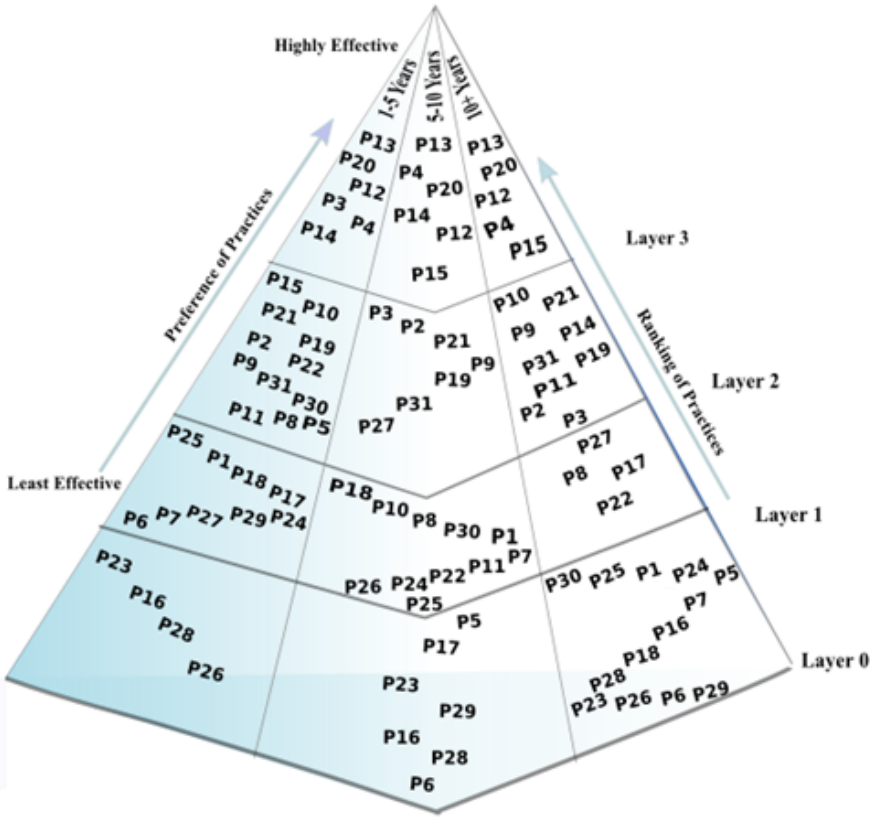

Figure 6.10: Hierarchy of practices based on the number of years in OSS

6.3.3 Ranking based on the Number of Years in Programming

The opinion on the effectiveness of practices is evaluated for contributors in OSS with varying number of years in computer programming based on the value of mean, median, SD, and MAD is depicted in Figure 6.11. The population is categorised into four groups by years: 1-2, 2-5, 5-10, and 10+. The difference of opinion on the effectiveness of practices can be observed in Figure 6.11.

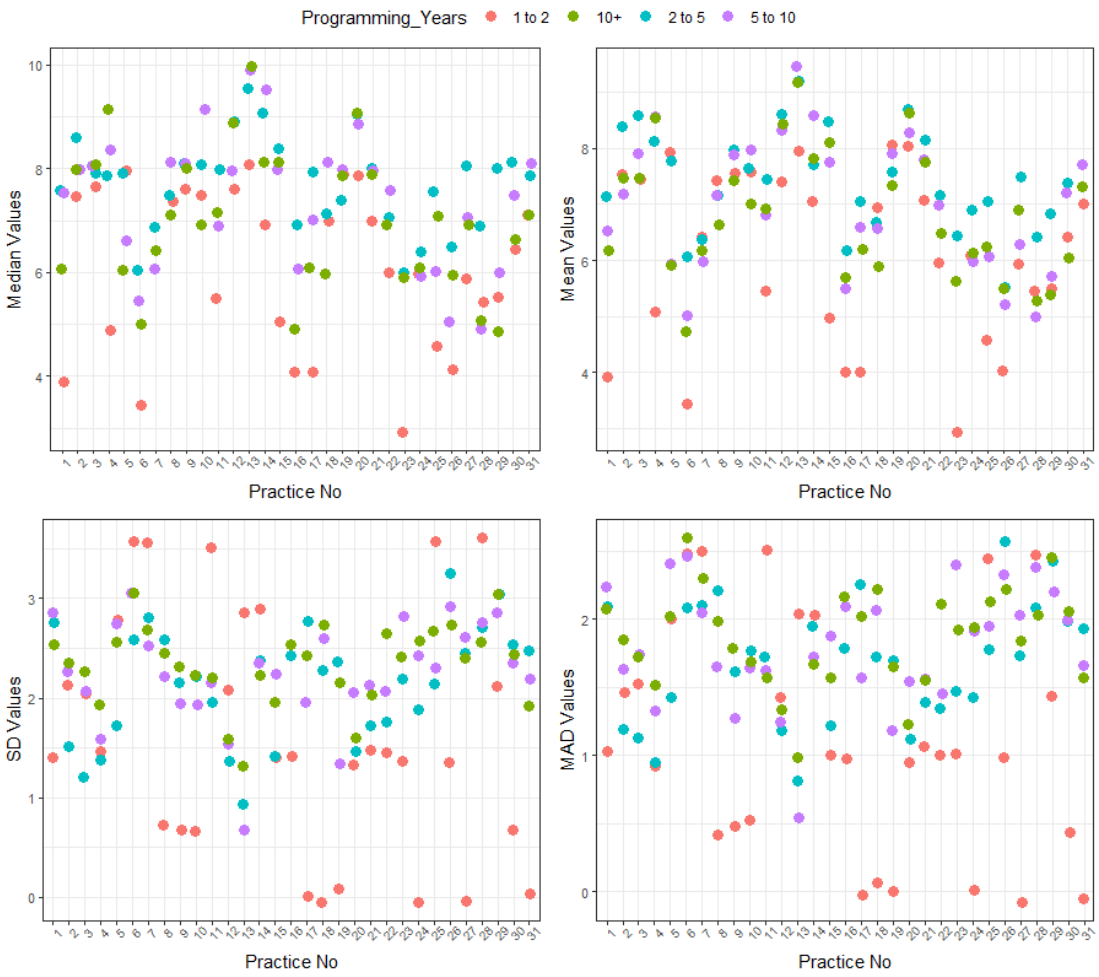

Figure 6.11: The reported effectiveness of practices based on number of years in computer programming

The ranking of practice effectiveness, categorised by the number of years in computer programming, is presented in Tables 6.10, 6.11, 6.12, 6.13.

Table 6.10: Categorical ranking of practices based on 1-2 numbers of years in programming

Practice Rank No.	**Calculations 1-2 Years in Computer Programming**				**Practice No.**
	Mean	Median	SD	MAD	
1	8	8	0	0	**P19**
2	8	8	1.414	1	**P20**
3	8	8	2.828	2	**P5**
4	8	8	2.828	2	**P13**
5	7.5	7.5	0.707	0.5	**P8**
6	7.5	7.5	0.707	0.5	**P9**
7	7.5	7.5	0.707	0.5	**P10**
8	7.5	7.5	2.121	1.5	**P2**
9	7.5	7.5	2.121	1.5	**P3**
10	7.5	7.5	2.121	1.5	**P12**
11	7	7	0	0	**P18**
12	7	7	0	0	**P31**
13	7	7	1.414	1	**P21**
14	7	7	2.828	2	**P14**
15	6.5	6.5	0.707	0.5	**P30**
16	6.5	6.5	3.536	2.5	**P7**
17	6	6	0	0	**P24**
18	6	6	0	0	**P27**
19	6	6	1.414	1	**P22**
20	5.5	5.5	2.121	1.5	**P29**
21	5.5	5.5	3.536	2.5	**P11**
22	5.5	5.5	3.536	2.5	**P28**
23	5	5	1.414	1	**P4**
24	5	5	1.414	1	**P15**
25	4.5	4.5	3.536	2.5	**P25**
26	4	4	0	0	**P17**
27	4	4	1.414	1	**P1**
28	4	4	1.414	1	**P16**
29	4	4	1.414	1	**P26**
30	3.5	3.5	3.536	2.5	**P6**
31	3	3	1.414	1	**P23**

Table 6.11: Categorical ranking of practices based on 2-5 numbers of years in programming

Practice Rank No.	Calculations 2-5 Years in Computer Programming				Practice No.
	Mean	Median	SD	MAD	
1	9.182	9.5	1.006	0.818	P13
2	8.727	9	1.518	1.099	P20
3	8.682	9	1.393	1.132	P12
4	8.636	8	1.177	1.058	P3
5	8.5	8.5	1.472	1.136	P15
6	8.409	8.5	1.563	1.136	P2
7	8.136	8	1.726	1.331	P21
8	8.091	8	1.411	0.95	P4
9	7.909	8	2.091	1.661	P9
10	7.727	8	1.804	1.479	P5
11	7.727	9	2.354	1.868	P14
12	7.727	8	2.529	1.893	P31
13	7.591	8	2.175	1.81	P10
14	7.591	7.5	2.404	1.773	P19
15	7.409	8	2.364	1.736	P27
16	7.409	8	2.482	1.971	P30
17	7.364	8	2.013	1.785	P11
18	7.136	7	1.781	1.343	P22
19	7.136	7.5	2.21	1.864	P25
20	7.136	7.5	2.731	2.136	P1
21	7.091	7.5	2.617	2.182	P8
22	7	8	2.845	2.182	P17
23	6.909	8	2.959	2.388	P29
24	6.864	6.5	1.935	1.5	P24
25	6.727	7	2.229	1.777	P18
26	6.5	6	2.177	1.545	P23
27	6.409	7	2.649	2.099	P28
28	6.409	7	2.737	2.062	P7
29	6.227	7	2.349	1.843	P16
30	6.045	6	2.627	2.054	P6
31	5.545	6.5	3.203	2.529	P26

Table 6.12: Categorical ranking of practices based on 5-10 numbers of years in programming

Practice Rank No.	Calculations 5-10 Years in Computer Programming				Practice No.
	Mean	Median	SD	MAD	
1	9.5	10	0.707	0.588	P13
2	8.529	8.5	1.581	1.353	P4
3	8.529	9.5	2.351	1.664	P14
4	8.382	8	1.557	1.251	P12
5	8.353	9	2.13	1.602	P20
6	8.029	9	2.007	1.557	P10
7	7.882	8	1.996	1.343	P9
8	7.882	8	2.157	1.785	P3
9	7.853	8	2.062	1.476	P21
10	7.824	8	1.424	1.1	P19
11	7.824	8	2.329	1.817	P15
12	7.735	8	2.287	1.637	P31
13	7.147	8	2.204	1.651	P2
14	7.147	8	2.258	1.709	P8
15	7.147	7.5	2.426	1.971	P30
16	6.971	7.5	2.096	1.512	P22
17	6.765	7	2.203	1.616	P11
18	6.559	8	2.676	2.133	P18
19	6.5	7	1.911	1.588	P17
20	6.441	7.5	2.776	2.22	P1
21	6.353	7	2.581	2.114	P27
22	6.029	6	2.329	1.913	P25
23	5.941	6.5	2.696	2.356	P5
24	5.912	6	2.417	1.917	P24
25	5.912	6	2.539	2.104	P7
26	5.794	6	2.89	2.22	P29
27	5.618	6	2.829	2.355	P23
28	5.471	6	2.585	2.09	P16
29	5.176	5	2.844	2.266	P26
30	5.029	5.5	3.02	2.5	P6
31	4.941	5	2.817	2.308	P28

Table 6.13: Categorical ranking of practices based on 10+ numbers of years in programming

Practice Rank No.	Calculations 10+ years in Computer Programming				Practice No.
	Mean	Median	SD	MAD	
1	9.265	10	1.4	0.952	P13
2	8.632	9	1.515	1.237	P20
3	8.618	9	1.901	1.465	P4
4	8.426	9	1.66	1.389	P12
5	8.029	8	1.877	1.504	P15
6	7.838	8	2.243	1.705	P14
7	7.75	8	2.105	1.632	P21
8	7.529	8	2.282	1.758	P3
9	7.471	8	2.397	1.901	P2
10	7.441	8	2.275	1.781	P9
11	7.412	8	2.089	1.616	P19
12	7.353	7	1.914	1.599	P31
13	7.103	7	2.234	1.73	P10
14	6.912	7	2.218	1.568	P11
15	6.897	7	2.351	1.869	P27
16	6.632	7	2.4	1.922	P8
17	6.515	7	2.623	2.086	P22
18	6.265	6	2.447	1.965	P17
19	6.206	7	2.629	2.146	P25
20	6.132	6.5	2.515	2.015	P30
21	6.132	6	2.568	2.048	P1
22	6.118	6.5	2.767	2.353	P7
23	6.059	6	2.503	1.943	P24
24	6	6	2.504	2	P5
25	5.926	6	2.788	2.267	P18
26	5.618	6	2.504	1.986	P23
27	5.618	5	2.557	2.136	P16
28	5.574	6	2.75	2.216	P26
29	5.338	5	3.04	2.535	P29
30	5.294	5	2.545	1.958	P28
31	4.735	5	3.065	2.67	P6

The pyramid in 6.12 depicts the hierarchy of the practice effectiveness as reported and based on the number of years contributing to OSS projects. The group of practices considered highly effective by OSS Contributors appear in the topmost layer of the pyramid in 6.12. The practices that considered least effective by OSS contributors cluster at the base of the pyramid.

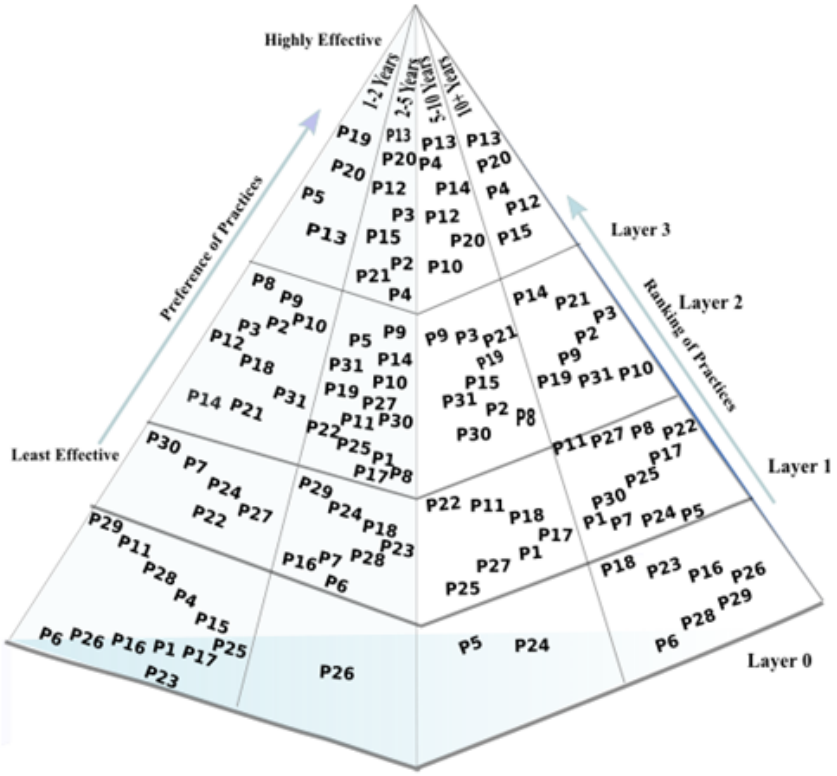

Figure 6.12: Hierarchy of practices based on the number of years in computer programming

6.4 Ranking based on role type in OSS

In OSS projects, contributors may perform a variety of tasks simultaneously. A ranking of practices based upon on their reported effectiveness by role type gives an insight on the preference of certain practices over others. In this section, responses from contributors in nine different OSS project role types are analysed. The observed values of mean, median, SD, and MAD for each role are given in Figures 6.13, 6.14, 6.15, and 6.16 respectively. The overall ranking of practices can be viewed for each in the following sub sections.

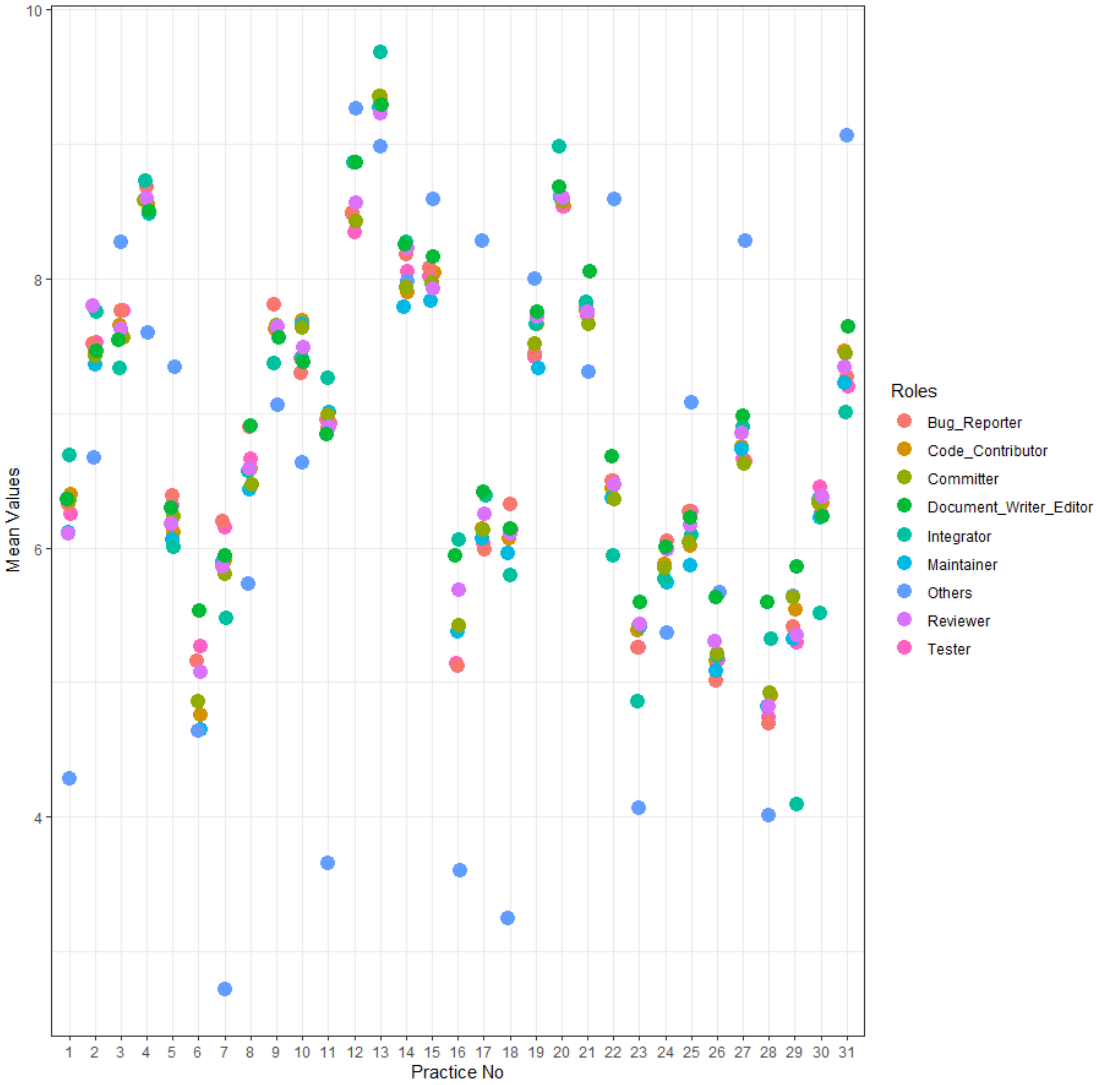

Figure 6.13: Mean values by role type

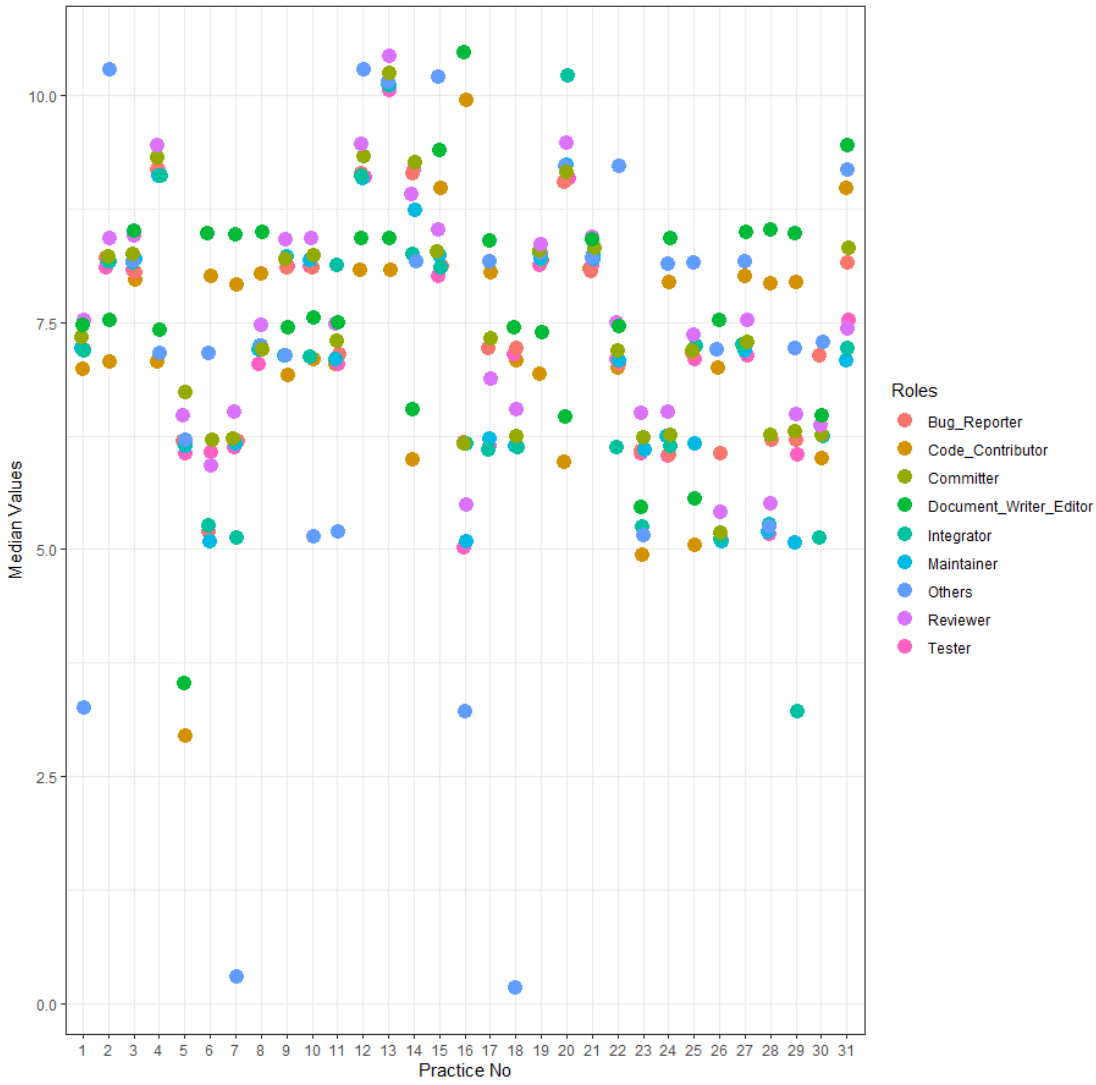

Figure 6.14: Median values by role type

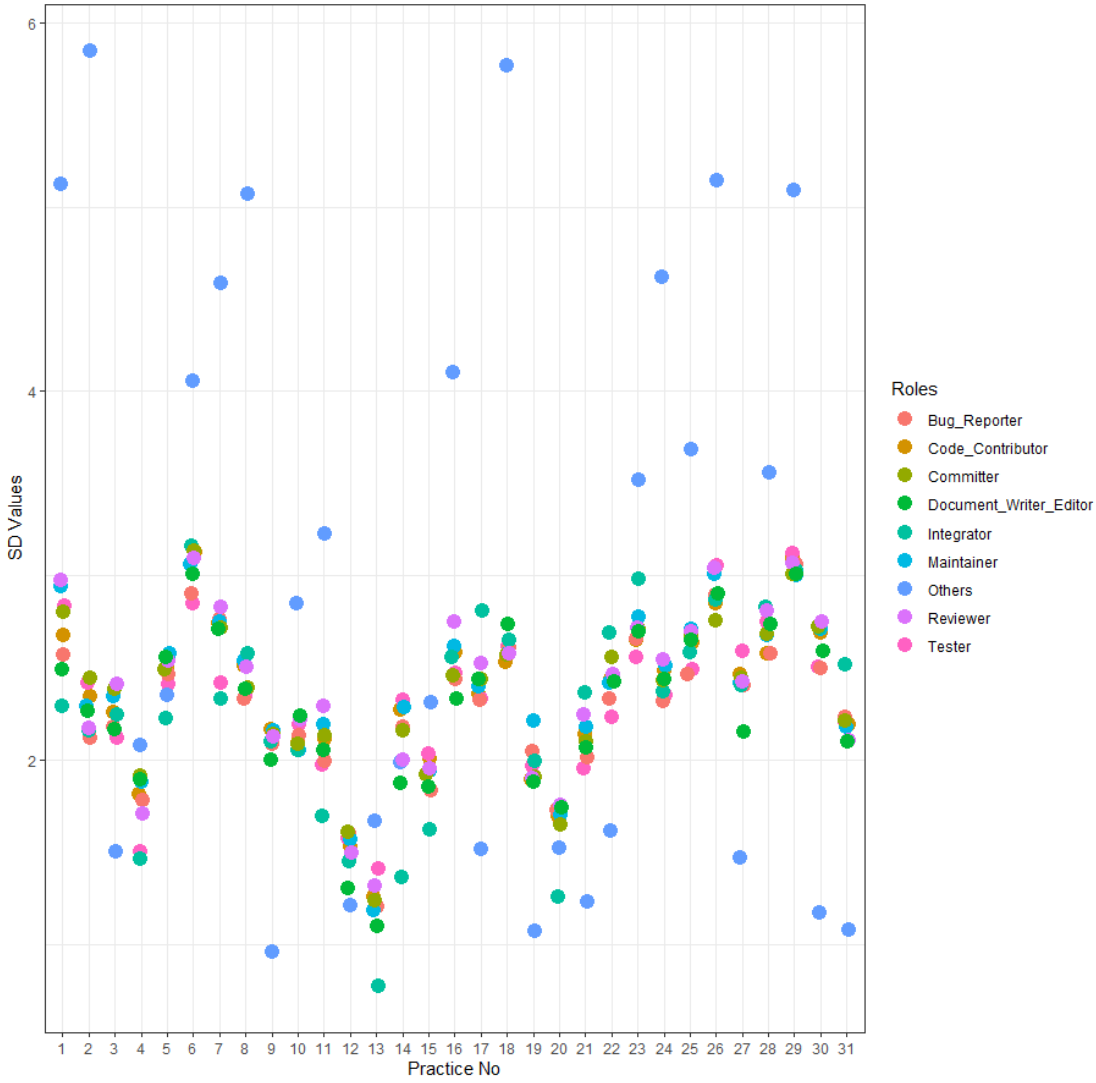

Figure 6.15: SD values by role type

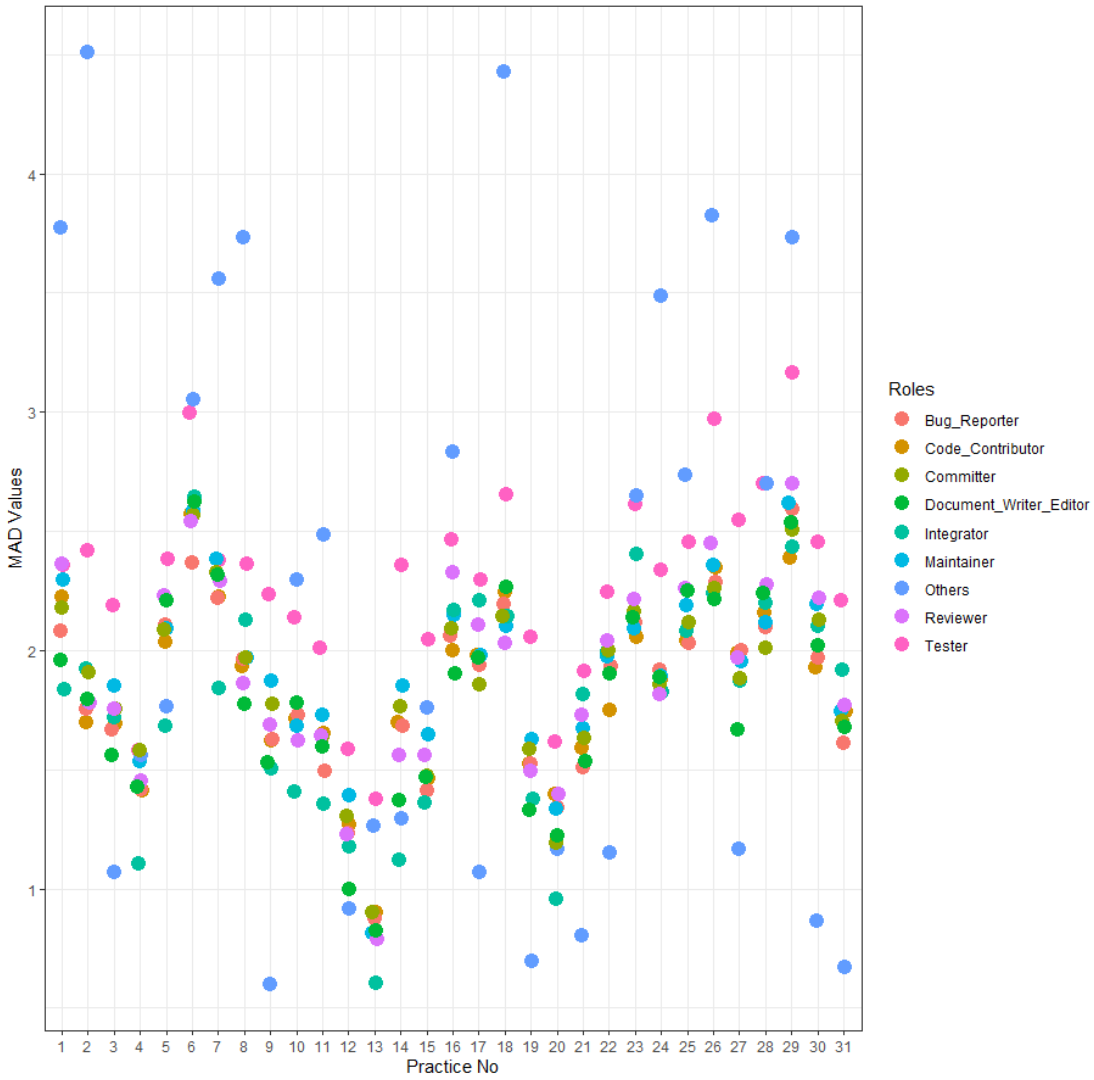

Figure 6.16: MAD values by role type

6.4.1 Bug Reporter

The ranking of practice effectiveness based on the role of OSS bug reporter is identified in Table 6.14 and demonstrates that P13, P4, P20, P12, P14, and P15 are considered to be the most effective practices by OSS project bug reporters, with P16, P6, P26, and P28 considered the least effective.

Table 6.14: Ranking of practices based on bug reporters in OSS projects

Practice Rank No.	Calculation Bug Reporter				Practice No.
	Mean	Median	SD	MAD	
1	9.235	10	1.163	0.816	P13
2	8.647	9	1.774	1.400	P4
3	8.544	9	1.752	1.348	P20
4	8.515	9	1.519	1.286	P12
5	8.176	9	2.265	1.750	P14
6	8.059	8	1.874	1.448	P15
7	7.824	8	2.113	1.656	P9
8	7.75	8	2.050	1.549	P21
9	7.706	8	2.091	1.618	P3
10	7.544	8	2.166	1.698	P2
11	7.515	8	2.047	1.545	P19
12	7.324	8	2.172	1.759	P10
13	7.191	8	2.151	1.686	P31
14	6.853	7	2.066	1.547	P11
15	6.824	7	2.401	1.916	P8
16	6.603	7	2.459	1.954	P27
17	6.515	7	2.388	1.858	P22
18	6.397	7	2.620	2.151	P18
19	6.382	6	2.422	2.015	P5
20	6.309	7	2.531	2.056	P25
21	6.279	7	2.620	2.119	P1
22	6.265	6	2.726	2.279	P7
23	6.191	7	2.540	2.051	P30
24	6.029	7	2.373	1.918	P17
25	5.971	6	2.385	1.847	P24
26	5.353	6	3.021	2.509	P29
27	5.191	6	2.616	2.079	P23
28	5.118	6	2.470	2.025	P16
29	5.103	5	2.934	2.450	P6
30	5.074	6	2.813	2.280	P26
31	4.721	6	2.654	2.104	P28

6.4.2 Code Contributor

The role of code contributors, which is prominent in the data collected (17%), identifies P13, P20, P12, P4, P15 and P14 as the most effective practices. The least effective practices for code contributors are P29, P16, P26, P28, and P6. Table 6.15 details the values of mean, media, SD, and MAD utilised in the ranking of practices, as found to be effective by code contributors.

Table 6.15: Ranking of practices based on code contributors in OSS projects

Practice Rank No.	Calculation Code Contributor				Practice No.
	Mean	Median	SD	MAD	
1	9.288	10	1.210	0.854	P13
2	8.568	9	1.696	1.314	P20
3	8.448	9	1.589	1.328	P12
4	8.432	9	1.784	1.432	P4
5	7.992	8	1.974	1.530	P15
6	7.976	9	2.298	1.760	P14
7	7.848	8	2.012	1.523	P21
8	7.8	8	2.110	1.616	P3
9	7.624	8	2.147	1.655	P9
10	7.568	8	1.977	1.497	P19
11	7.528	8	2.231	1.725	P2
12	7.496	8	2.108	1.668	P31
13	7.448	8	2.176	1.769	P10
14	6.92	7	2.191	1.640	P11
15	6.856	7	2.385	1.908	P8
16	6.824	7	2.417	1.916	P27
17	6.736	7	2.349	1.822	P22
18	6.608	7	2.501	2.017	P30
19	6.432	7	2.394	1.951	P17
20	6.328	7	2.648	2.144	P1
21	6.288	6	2.514	2.087	P5
22	6.272	7	2.500	2.044	P25
23	6.264	7	2.658	2.161	P18
24	6.144	6	2.378	1.832	P24
25	6.112	7	2.692	2.255	P7
26	5.728	6	2.563	2.034	P23
27	5.704	6	2.987	2.480	P29
28	5.664	6	2.524	2.088	P16
29	5.424	6	2.835	2.302	P26
30	5.376	6	2.663	2.133	P28
31	4.984	5	2.973	2.499	P6

6.4.3 Maintainer

Table 6.16 depicts the practice ranking based on the reported effectiveness by OSS project maintainers who identify P13, P20, P4, and P12 as the most effective practices and P16, P29, P26, P28, and P6 as least effective.

Table 6.16: Ranking of practices based on maintainers in OSS projects

Practice Rank No.	Calculation Maintainer				Practice No.
	Mean	Median	SD	MAD	
1	9.326	10	1.268	0.835	P13
2	8.663	9	1.686	1.295	P20
3	8.446	9	1.906	1.54	P4
4	8.413	9	1.618	1.373	P12
5	7.902	8	1.922	1.569	P15
6	7.837	8.5	2.317	1.836	P14
7	7.728	8	2.194	1.716	P21
8	7.62	8	2.117	1.664	P10
9	7.587	8	2.302	1.833	P3
10	7.565	8	2.235	1.784	P9
11	7.435	8	2.373	1.869	P2
12	7.413	8	2.129	1.62	P19
13	7.261	7	2.253	1.794	P31
14	7.033	7	2.27	1.664	P11
15	6.641	7	2.466	1.943	P27
16	6.489	7	2.496	2.034	P8
17	6.348	7	2.46	1.947	P22
18	6.293	6	2.638	2.145	P30
19	6.141	6	2.425	1.942	P17
20	6.13	7	2.883	2.361	P1
21	6.065	6	2.506	2.032	P5
22	6	6	2.648	2.13	P18
23	5.913	6	2.638	2.165	P25
24	5.859	6	2.772	2.318	P7
25	5.815	6	2.524	1.926	P24
26	5.467	6	2.703	2.164	P23
27	5.337	5	2.603	2.098	P16
28	5.304	5	3.052	2.557	P29
29	5.087	5	2.926	2.334	P26
30	4.772	5	2.648	2.104	P28
31	4.728	5	3.049	2.584	P6

6.4.4 Reviewer

Of the survey participants, 11% report being OSS project reviewers. Among OSS project reviewers, P13, P20, P4, P12, and P14 are considered the most effective practices, with P23, P29, P26, P6, and P28 the least effective. Table 6.17 depicts the practice effectiveness ranking as reported by OSS projct reviewers.

Table 6.17: Ranking of practices based on reviewers in OSS projects

Practice Rank No.	Calculation Reviewer				Practice No.
	Mean	Median	SD	MAD	
1	9.3	10	1.287	0.875	P13
2	8.65	9	1.692	1.321	P20
3	8.575	9	1.777	1.449	P4
4	8.513	9	1.458	1.237	P12
5	8.163	8.5	1.996	1.538	P14
6	7.938	8	1.964	1.6	P15
7	7.75	8	2.19	1.681	P21
8	7.725	8	1.955	1.487	P19
9	7.725	8	2.25	1.744	P2
10	7.638	8	2.189	1.683	P9
11	7.575	8	2.326	1.821	P3
12	7.413	8	2.139	1.667	P10
13	7.4	7	2.191	1.735	P31
14	6.888	7	2.278	1.699	P11
15	6.788	7	2.401	1.891	P27
16	6.613	7	2.473	1.94	P8
17	6.425	7	2.54	1.976	P22
18	6.313	6	2.679	2.17	P30
19	6.288	6.5	2.557	2.088	P17
20	6.25	6	2.607	2.163	P5
21	6.2	7	2.674	2.175	P25
22	6.163	7	2.897	2.371	P1
23	6.15	6	2.634	2.079	P18
24	6	6	2.47	1.9	P24
25	5.9	6	2.804	2.31	P7
26	5.625	5	2.716	2.241	P16
27	5.363	6	2.715	2.153	P23
28	5.35	6	3.098	2.616	P29
29	5.213	5	2.971	2.384	P26
30	5.025	5.5	3.134	2.625	P6
31	4.8	5	2.794	2.265	P28

6.4.5 Committer

The contributors who have the right to commit code to OSS repositories represent 13% of the total population of contributors in the data collection. Table 6.18 depicts the ranking of practices based on the opinion of committers on the effectiveness of certain practices over other practices. Accordingly, the five top most effective practices are P13, P20, P4, P12, and P15, Similarly the

least effective practices reported for the role of committer are P23, P16, P26, P28, and P6.

Table 6.18: Ranking of practices based on committers in OSS projects

Practice Rank No.	Calculations Committer				Practice No.
	Mean	Median	SD	MAD	
1	9.326	10	1.16	0.809	P13
2	8.62	9	1.741	1.281	P20
3	8.543	9	1.833	1.502	P4
4	8.467	9	1.272	1.025	P12
5	8.011	8	1.845	1.485	P15
6	7.957	9	1.936	1.452	P14
7	7.728	8	2.038	1.485	P21
8	7.696	8	2.045	1.519	P9
9	7.641	8	2.105	1.617	P3
10	7.598	8	2.235	1.778	P10
11	7.576	8	1.844	1.337	P19
12	7.38	8	2.282	1.83	P2
13	7.38	8	2.075	1.701	P31
14	6.913	7	2.029	1.532	P11
15	6.707	7	2.236	1.683	P27
16	6.543	7	2.379	1.832	P8
17	6.391	7	2.379	1.854	P22
18	6.37	6	2.6	2.054	P30
19	6.315	7	2.487	1.978	P1
20	6.174	6.5	2.618	2.167	P5
21	6.109	6	2.754	2.202	P18
22	6.065	7	2.451	2	P17
23	5.978	7	2.686	2.209	P25
24	5.891	6	2.789	2.322	P7
25	5.826	6	2.468	1.971	P24
26	5.543	6	3.015	2.493	P29
27	5.446	6	2.731	2.162	P23
28	5.435	6	2.395	1.896	P16
29	5.207	5	2.834	2.29	P26
30	4.957	6	2.76	2.256	P28
31	4.772	6	3.085	2.573	P6

6.4.6 Document Writer and Editor

The role of document writer and editor constitutes 9% of the total population in data collection of OSS projects. Table 6.19 depicts the ranking obtained from

document writers and editors based on their responses regarding the effectiveness of practices. The top five practices for document writers are P13, P12, P20, P4, and P14, with P29, P23, P28, P26, and P6 being reported as the least effective practices within the document writer and editor OSS role type.

Table 6.19: Ranking of practices based on document writers and editors in OSS Projects

Practice Rank No.	Calculations Document Writer				Practice No.
	Mean	Median	SD	MAD	
1	9.379	10	1.16	0.809	P13
2	8.833	9	1.272	1.025	P12
3	8.727	9	1.741	1.281	P20
4	8.47	9	1.833	1.502	P4
5	8.288	9	1.936	1.452	P14
6	8.167	8	1.845	1.485	P15
7	8	8	2.038	1.485	P21
8	7.788	8	1.844	1.337	P19
9	7.591	8	2.045	1.519	P9
10	7.591	8	2.075	1.701	P31
11	7.591	8	2.105	1.617	P3
12	7.53	8	2.282	1.83	P2
13	7.333	8	2.235	1.778	P10
14	7.015	7	2.236	1.683	P27
15	6.818	7	2.379	1.832	P8
16	6.773	7	2.029	1.532	P11
17	6.591	7	2.379	1.854	P22
18	6.5	7	2.451	2	P17
19	6.409	7	2.487	1.978	P1
20	6.288	6	2.6	2.054	P30
21	6.273	7	2.686	2.209	P25
22	6.227	6.5	2.618	2.167	P5
23	6.167	6	2.754	2.202	P18
24	6.03	6	2.468	1.971	P24
25	5.985	6	2.395	1.896	P16
26	5.864	6	2.789	2.322	P7
27	5.788	6	3.015	2.493	P29
28	5.667	6	2.731	2.162	P23
29	5.636	6	2.76	2.256	P28
30	5.576	5	2.834	2.29	P26
31	5.53	6	3.085	2.573	P6

6.4.7 Tester

Testers in the collected data from OSS projects constitute 17% of the population, which is equivalent to that of code contributors. The effectiveness of practices evaluated by the group of OSS project testers is shown in Table 6.20. The top most effective practices as reported by testers are P13, P4, P20, P12, and P14, with the least effective being P6, P16, P23, P26, and P28.

Table 6.20: Ranking of practices based on testers in OSS projects

Practice Rank No.	Calculations Testers				Practice No.
	Mean	Median	SD	MAD	
1	9.225	10	1.359	1.359	P13
2	8.575	9	1.565	1.565	P4
3	8.525	9	1.676	1.676	P20
4	8.388	9	1.619	1.619	P12
5	8.05	9	2.272	2.272	P14
6	7.975	8	2.081	2.081	P15
7	7.713	8	2.001	2.001	P21
8	7.7	8	2.155	2.155	P3
9	7.688	8	2.15	2.15	P9
10	7.5	8	2.328	2.328	P2
11	7.463	8	2.037	2.037	P19
12	7.375	8	2.149	2.149	P10
13	7.238	7.5	2.291	2.291	P31
14	7	7	1.955	1.955	P11
15	6.688	7	2.352	2.352	P8
16	6.575	7	2.594	2.594	P27
17	6.488	7	2.284	2.284	P22
18	6.463	7	2.526	2.526	P30
19	6.338	7	2.444	2.444	P25
20	6.263	6	2.412	2.412	P5
21	6.25	7	2.862	2.331	P1
22	6.2	7	2.679	2.679	P18
23	6.075	6	2.453	2.453	P7
24	5.95	6	2.338	2.338	P17
25	5.925	6	2.391	2.391	P24
26	5.35	6	3.077	3.077	P29
27	5.275	6	2.938	2.938	P6
28	5.2	5	2.553	2.553	P16
29	5.2	6	2.572	2.572	P23
30	5.113	5	2.968	2.968	P26
31	4.788	5	2.722	2.722	P28

6.4.8 Integrator

Integrators constitute 4% of the total population in the collected data. The ranking in Table 20 gives the overall opinion on the effectiveness of practices as reported by integrators. In Table 6.21, the ranking for the most effective practices for integrators are P13, P20, P12, P4, and P14; and the least effective practices for OSS project integrators are P28, P26, P6, P23, and P29.

Table 6.21: Ranking of practices based on integrators in OSS projects

Practice Rank No.	Calculation Integrators				Practice No.
	Mean	Median	SD	MAD	
1	9.655	10	0.721	0.523	P13
2	9.034	10	1.267	0.999	P20
3	8.828	9	1.537	1.203	P12
4	8.759	9	1.38	1.18	P4
5	8.207	8	1.424	1.139	P14
6	7.966	8	1.679	1.417	P15
7	7.897	8	2.425	1.853	P21
8	7.69	8	2.206	1.838	P2
9	7.621	8	1.916	1.367	P19
10	7.379	7	2.145	1.522	P9
11	7.379	8	2.243	1.788	P3
12	7.345	7	2.075	1.46	P10
13	7.241	8	1.725	1.405	P11
14	7.034	7	2.442	1.971	P31
15	6.931	7	2.448	1.879	P27
16	6.655	7	2.303	1.853	P1
17	6.517	7	2.572	2.05	P8
18	6.414	6	2.835	2.278	P17
19	6.034	6	2.57	2.107	P16
20	6.034	7	2.598	2.169	P25
21	5.966	6	2.244	1.693	P5
22	5.931	6	2.604	2.017	P22
23	5.862	6	2.722	2.093	P18
24	5.69	6	2.301	1.838	P24
25	5.552	5	2.339	1.788	P7
26	5.483	5	2.707	2.05	P30
27	5.31	5	2.817	2.226	P28
28	5.241	5	2.786	2.214	P26
29	5.138	5	3.148	2.566	P6
30	4.793	5	3.028	2.464	P23
31	4.034	3	3.065	2.521	P29

6.4.9 Others

A very small number of contributors also selected the role of "other", where they mainly identified as community manager, and community supporter. The ranking on the effectiveness of practices for these role types is depicted in Table 6.22. P12, P31, P13, P20, and P22 are rated as the most effective practices in this cohort, with P28, P11, P16, P18, and P7 being the least effective.

Table 6.22: Ranking of practices based on others in OSS projects

Practice Rank No.	Calculation Others				Practice No.
	Mean	Median	SD	MAD	
1	9.333	10	1.155	0.889	P12
2	9	9	1	0.667	P31
3	9	10	1.732	1.333	P13
4	8.667	9	1.528	1.111	P20
5	8.667	9	1.528	1.111	P22
6	8.667	10	2.309	1.778	P15
7	8.333	8	1.528	1.111	P3
8	8.333	8	1.528	1.111	P17
9	8.333	8	1.528	1.111	P27
10	8	8	1	0.667	P19
11	8	8	2	1.333	P14
12	7.667	7	2.082	1.556	P4
13	7.333	8	1.155	0.889	P21
14	7.333	6	2.309	1.778	P5
15	7	7	1	0.667	P9
16	7	8	3.606	2.667	P25
17	6.667	5	2.887	2.222	P10
18	6.667	10	5.774	4.444	P2
19	6.333	7	1.155	0.889	P30
20	5.667	7	5.132	3.778	P8
21	5.667	7	5.132	3.778	P26
22	5.667	7	5.132	3.778	P29
23	5.333	8	4.619	3.556	P24
24	4.667	7	4.041	3.111	P6
25	4.333	3	5.132	3.778	P1
26	4	5	3.606	2.667	P23
27	4	5	3.606	2.667	P28
28	3.667	5	3.215	2.444	P11
29	3.667	3	4.041	2.889	P16
30	3.333	0	5.774	4.444	P18
31	2.667	0	4.619	3.556	P7

Table 6.23 summarises the ranking of PKR practices for all 9 roles including "Others".

Table 6.23: Ranking of practices based on nine roles in OSS projects

Practice Rank No.	Bug Reporter	Code Contributor	Maintainer	Reviewer	Committer	Document Writer/ Editor	Tester	Integrator	Others
1	P13	P13	P13	P13	P13	P13	P13	P13	P12
2	P4	P20	P20	P20	P20	P12	P4	P20	P31
3	P20	P12	P4	P4	P4	P20	P20	P12	P13
4	P12	P4	P12	P12	P12	P4	P12	P4	P20
5	P14	P15	P15	P14	P15	P14	P14	P14	P22
6	P15	P14	P14	P15	P14	P15	P15	P15	P15
7	P9	P21	P21	P21	P21	P21	P21	P21	P3
8	P21	P3	P10	P19	P9	P19	P3	P2	P17
9	P3	P9	P3	P2	P3	P9	P9	P19	P27
10	P2	P19	P9	P9	P10	P31	P2	P9	P19
11	P19	P2	P2	P3	P19	P3	P19	P3	P14
12	P10	P31	P19	P10	P2	P2	P10	P10	P4
13	P31	P10	P31	P31	P31	P10	P31	P11	P21
14	P11	P11	P11	P11	P11	P27	P11	P31	P5
15	P8	P8	P27	P27	P27	P8	P8	P27	P9
16	P27	P27	P8	P8	P8	P11	P27	P1	P25
17	P22	P22	P22	P22	P22	P22	P22	P8	P10
18	P18	P30	P30	P30	P30	P17	P30	P17	P2
19	P5	P17	P17	P17	P1	P1	P25	P16	P30
20	P25	P1	P1	P5	P5	P30	P5	P25	P8
21	P1	P5	P5	P25	P18	P25	P1	P5	P26
22	P7	P25	P18	P1	P17	P5	P18	P22	P29
23	P30	P18	P25	P18	P25	P18	P7	P18	P24
24	P17	P24	P7	P24	P7	P24	P17	P24	P6
25	P24	P7	P24	P7	P24	P16	P24	P7	P1
26	P29	P23	P23	P16	P29	P7	P29	P30	P23
27	P23	P29	P16	P23	P23	P29	P6	P28	P28
28	P16	P16	P29	P29	P16	P23	P16	P26	P11
29	P6	P26	P26	P26	P26	P28	P23	P6	P16
30	P26	P28	P28	P6	P28	P26	P26	P23	P18
31	P28	P6	P6	P28	P6	P6	P28	P29	P7

6.5 Qualitative Data on PKR Model Completeness

As noted previously, there was a survey question included regarding the completeness of the practice listing created and used in this research. This question was only completed by a small number of respondents, where the prevailing sentiment was that the survey included a "rather exhaustive list". There was also an expression on "Actually, I took some ideas to my job project, thanks for sending it to me!". There were also some other interesting responses to this question, including the suggestion that a practice should be included to "identify ... contributors and talk with them". This suggests that in some projects, the contributors may not be known and this is an aspect of OSS projects that may be worthy future extended research. There was also some disagreement on the need for a Code of Conduct on OSS projects, with one respondent suggesting that such Codes not be used at all and another stressing that they should be policed more vigilantly. This too can be examined in more detail in future work. The creation of a public changelog as a means to register contributors was also recommended by one respondent. As part of later work, these additional suggestions can be examined further with a view to extending the canonical model. However, there were no instances of overwhelming and frequent suggestions for new practices or extensions to existing practices, and therefore, the decision in the context of this research is not to include these additional concepts at this time.

6.6 Chapter Summary

In this chapter an analysis of the data collected in the survey was presented. Of the 132 responses obtained, a total of 126 were deemed valid for analysis. the profiling demonstrates that a broad spectrum of OSS contributors have participated in this study. The data collected is representative of various OSS contributors contributing in different roles, with varying levels of experience in programming ranging from relatively inexperienced to highly experienced. As a general observation, the data collected for most of the PKR practices on plotting is not normally distributed.

The values of mean, median, SD, and MAD are utilised to rank of practices into a hierarchy. A consolidated view of PKR practice ranking based on the preference of OSS contributors is presented by ranking pyramid, where practices with highest rankings are placed in the topmost layer 3 and practices with least ranking are placed in the layer 0. The categorical ranking of PKR practices is also performed utilising the data on contributors' experience profile including contributions to the number of OSS projects, number of years in OSS, role(s) adopted in OSS, and number of years in general programming. The outcome of the categorical ranking is also consolidated as ranking pyramids. Finally, the rankings of PKR practices are presented based on contributor's role in OSS projects.

Taking an overview of the initial ranking of practices in the population, it is evident that P13, P20, P12, P4, P19, P21, P15, P3, P11, P9 are valued relatively higher than other practices. Similarly, practices P23, P16, P26, P28, and P6 are least valued across the population.

In the following chapter 8, the outcomes of this analysis are further evaluated to highlight PKR practices preference trend among contributors with varying experience in OSS projects.

Chapter 7

Evaluation of Practices

The previous chapter discussed the details of the statistical analysis performed on the data collected from the OSS community survey. The OSS contributors evaluated practices for their effectiveness from highly effective to least effective. The responses obtained from OSS contributors after the analysis resulted in the ranking of practices on the spectrum of 1 to 31 based on the varying experience and role(s) of contributors in OSS projects.

In this chapter, the opinion of various sub-groups concerning the effectiveness of particular knowledge retention practice is discussed. The sub-groups of interest stem from the classification sought in the survey itself, namely: 1) the total number of OSS projects contributed to, 2) the total number of years contributing to OSS projects, 3) the total number of years in computer programming, and 4) the role(s) performed in OSS project(s). The evaluation of disparity of reported practice effectiveness among OSS contributors is based on the assessment of mean, median, SD, and MAD values.

Section 7.1 evaluates the practice preference by contributors with an experience on certain number of OSS projects. Section 7.2 articulates the discussion on the practice preference of contributors categorised by the number of years working in OSS. Section 7.3 discusses the practice preference of contributors categorised on the number of years in computer programming. The discussion on the practice preference by contributors performing different roles in OSS follows in Section 7.4.

7.1 Evaluating Practice Preference – Number of OSS projects

This section elaborates on the reported effectiveness of knowledge retention practices evident by their acquired ranking from contributors with an experience on certain number of OSS projects. Figure 7.1 highlights the mean, median, SD, and MAD values from three categories of contributors for 31 practices.

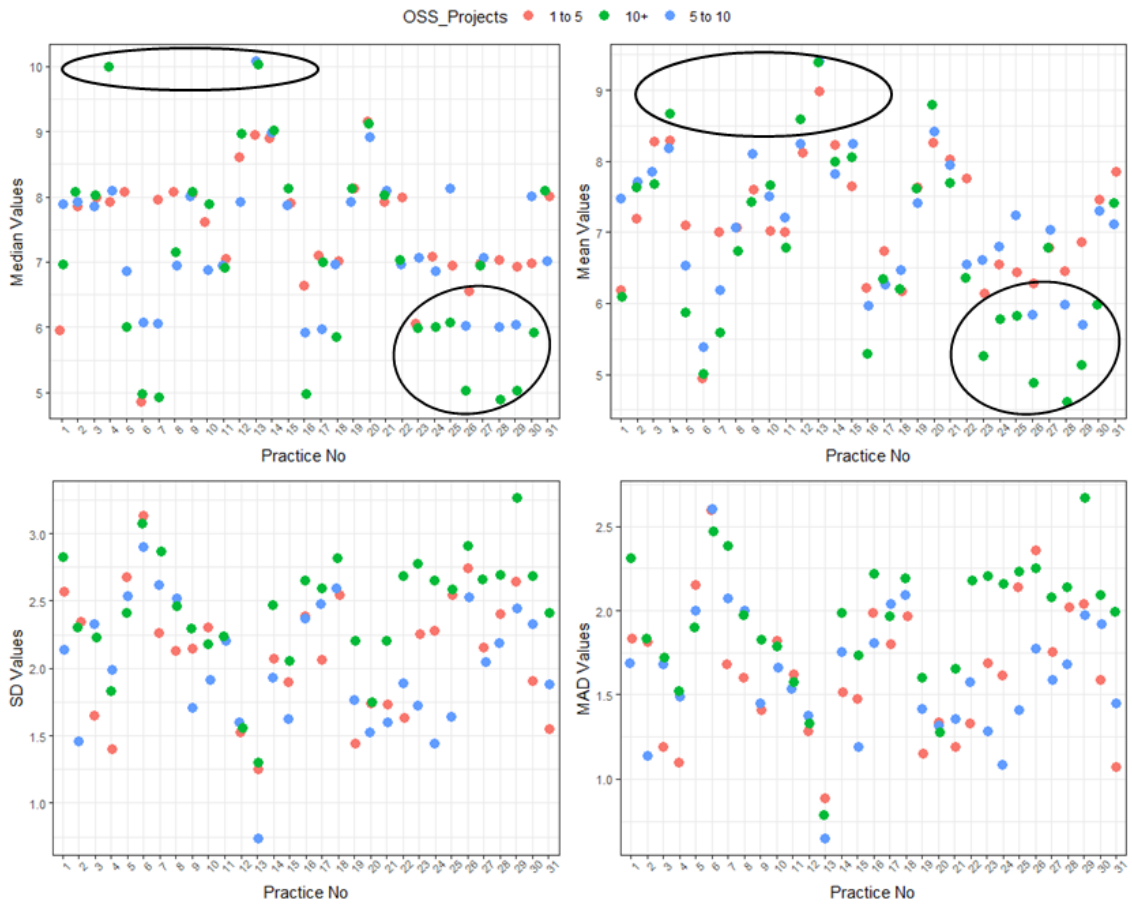

Figure 7.1: The practice preference for contributors with experience based on the number of OSS projects

The most interesting aspect of Figure 7.1 is the observation that there is a tendency for those with greater experience levels to downgrade the perceived effectiveness of certain practices, while at the same time these same participants are tending to place heightened importance on other practices. This is

particularly evident for P16 (Establish core to non-core knowledge sharing practices (e.g. interviews)). Practices P23 - 26 (Proactive assignment of varying tasks to non-core contributors; Advocate diversification of core contributor specialisation; Distribute project leadership to the wider support community when primary leaders are absent; Document role progression process (e.g. from contributor to integrator to maintainers), P28 (Establish training mechanisms (e.g. on-the-job training and shadowing)) and P29 (Create a knowledge contribution recognition program (e.g. a reward structure)), which are all reported to have relatively less effectiveness among those with the highest number of OSS projects.

There are some considerable grounds for concern in some of this information; for example, it suggests that the most experienced OSS contributors in terms of numbers of projects worked on do not value highly some core KR practices that are focused on assisting those with less experience/ knowledge on projects, including establishing non-core knowledge sharing practices, establishing training mechanisms and creating a knowledge contribution recognition programme (all of which are ranked lowly). The reasons for this might be that such practices in fact create work for the more experienced contributors, and more of the type of work that perhaps they are not inclined towards. Perhaps there is an expectation among seasoned OSS campaigners that individuals should be able to get up to speed themselves. This might be applicable to some extent, but the greater resilience of larger and growing projects must, over time, depend on continuing to attract contributors. In this respect, there are perhaps some knowledge sharing frailties in the more experienced OSS contributors. And this conflicts somewhat with the reported effectiveness of the practice of promoting an altruistic knowledge sharing culture in OSS projects (P12), which is rated as highly effective by the most experienced contributors.

From this data, one possible interpretation is that the most experienced contributors value altruism and knowledge exchange, but are to some extent

restrained when it comes to actually supporting these activities personally. Furthermore, this data suggests that those with lower levels of OSS project experience crave greater recognition and importance in OSS projects, while those who have already attained these objectives are not particularly inclined to spend time helping others to supplant them. However, in respect of P4 (Ensure the presence of testing artefacts (e.g. unit tests, test scripts, test cases)), we can see that those who have worked on greater numbers of projects value this practice very highly, and more highly than those of less experience. This suggests that testing artefacts are critically important to OSS projects, and the voice of experience recognises this point.

Interestingly in this case is the observation that the better the testing infrastructure on a project, there is perhaps a reduced need for other communications. This is tricky proposition to fully evaluate within the limited context of this research but one possible interpretation is that those who have worked on many projects have formed the view that the testing artefacts are highly important, and one can postulate good reasons for this in a community where contributors are transient and not bound to the project. At least the presence of testing artefacts can guard against the impact of discontinuity of contribution from certain contributors. As a basic observation: those who have worked on many projects would appear to be suggesting that it is much more important to get the testing artefacts from the contributors than to promote communication and training in the project. This observation should not be casually overlooked, it is perhaps a fundamental constraint on OSS projects that because of the transient nature of the contributor population, there is insufficient time or opportunity for training and various other forms of knowledge exchange, and therefore, getting the testing artefacts in place holds great value for the project.

Figure 7.2 depicts the ranking of the PKR practices based on the practice preference of OSS contributors with varying experience in OSS projects. In

Figure 7.2, further interesting insights are evident in the data based on experience in higher numbers of OSS projects. For example, practice P1 (encourage pair programming and shared code ownership), is most highly recommended by contributors with middle level of experience (5-10) and least preferred by contributors with 1-5 OSS projects. The reason might be that contributors' with 5-10 projects are knowledgeable and skilled to perform tasks on the project independently and collaborations can slow their work progress, while for inexperienced contributors (1-5 projects), this group may not yet be ready to collaborate because they are in the learning phase.

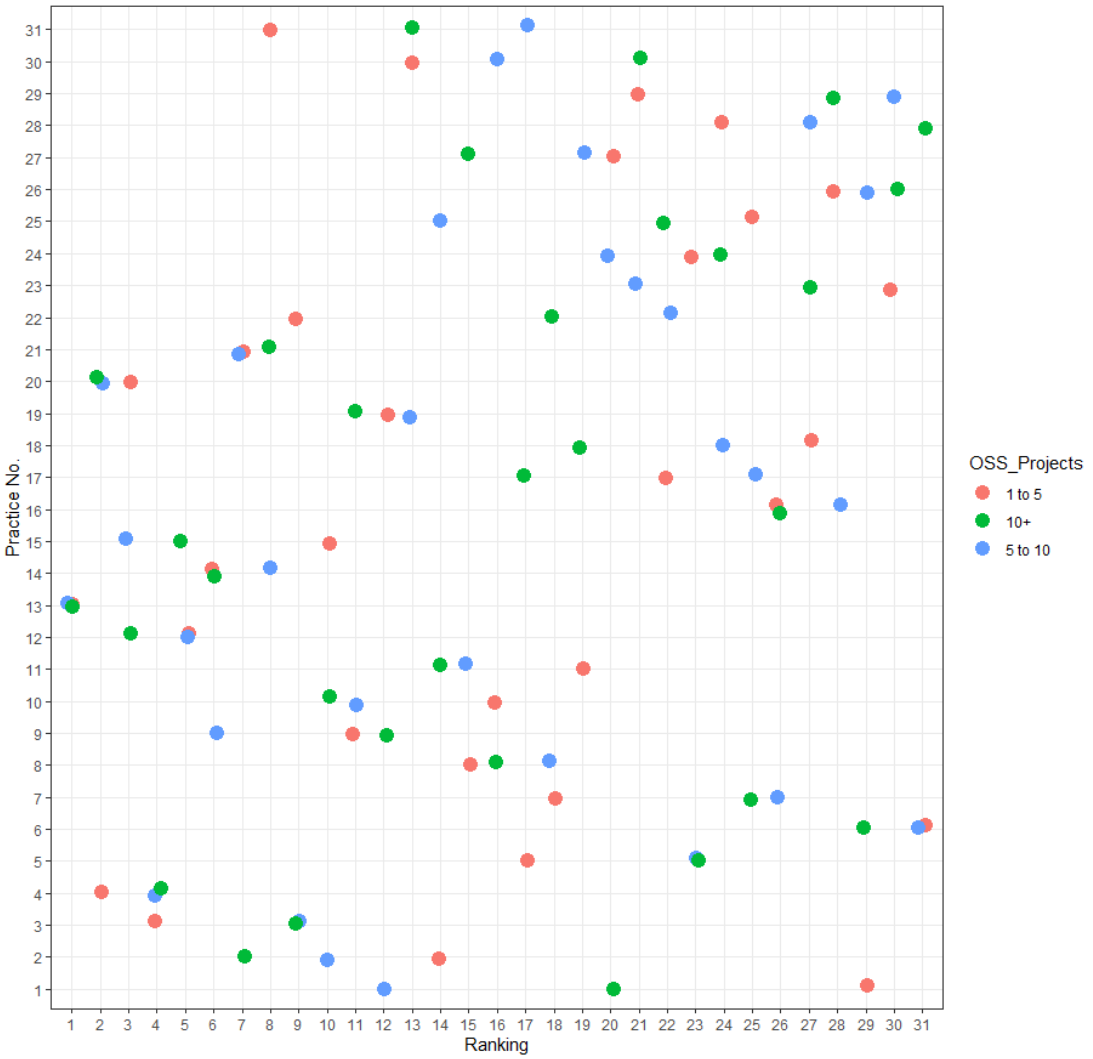

Figure 7.2: Ranking of practices based on contributors with an experience of 1-5, 5-10, 10+ in OSS projects

In Figure 7.2, practices P7, P22 and P31 are all interesting under this

evaluation because relatively inexperienced contributors (in terms of experience with numbers of OSS projects) have ranked these practices are being relatively more effective than those with greater numbers of OSS projects. P7 is concerned with having explicit code reviews and setting a maximum time limit for obtaining code review feedback. One interpretation here is that relatively inexperienced contributors seek timely feedback on their code contributions, perhaps as a mechanism to learn and grow, but perhaps also in an effort to integrate into the OSS project more smoothly and quickly. In contrast, more experienced contributors, whose time may be required in order to perform reviews, may place higher importance on other activities and, for example, may have a preference for actually coding over performing code reviews. This interpretation, if valid, would lend weight to the broader conclusion that those who are more experienced are perhaps more capable of getting up to speed, and prefer not to be too heavily engaged in knowledge dissemination activities, be they code reviews, or training, or other communication.

Relative newcomers are in contrast demonstrating a strong appetite to be involved in these same activities. This all falls back to the stakeholder objectives and it may be the case that those with less experience want more input from those of greater experience, but those with greater experience prefer to be doing what the prefer (e.g. coding) than helping less experienced staff get up to speed. This proposition is intuitively appealing but raised some issues for OSS projects as it may consolidate the major knowledge of a project in the minds of a relatively small number of contributors over time, and this may lessen the resilience and growth potential of the OSS project as a whole. Or, it may be that some very experienced individuals simply want to work on projects with other very experienced, self-motivated and self-starting individuals. The observation that P22 (identify successors for key code contributors) is ranked quite highly by those with less OSS project experience – having worked on fewer projects – and relatively lowly by those who have worked on many projects,

further suggest that relatively inexperienced individuals seek to become more experienced. They also, it would appear, seek to have their work explicitly recognised as evidenced by the relatively high ranking of P31 (promote a culture of appreciation for knowledge contributors (e.g. words of appreciation through blogs and newsletters)). Where work is more recognised, one's profile is raised, and one's opportunities increase. So, perhaps there is some quite basic natural competition phenomenon at work in this respect.

Generally speaking, for the majority of the other practices, the distance between the rankings and the means/medians is not particularly large and therefore, all that can be said for these other practices is that at a general level, participants tend to largely agree on the importance of these other practices irrespective of the number of OSS projects they have worked on.

7.2 Evaluating Practice Preference - Number of Years in OSS

In this section, the preference disparity of practices is discussed for contributors who have been contributing to OSS projects for some time (as opposed to the previous section which examined the total number of projects worked and not the number of elapsed years over which a contributor has been working on OSS projects). The contributors experience is presented in three categories, 1-5, 5-10, and 10+ years in OSS projects. The preference trend among three categories of contributors, based upon the number of years in OSS project is depicted in Figure 7.3.

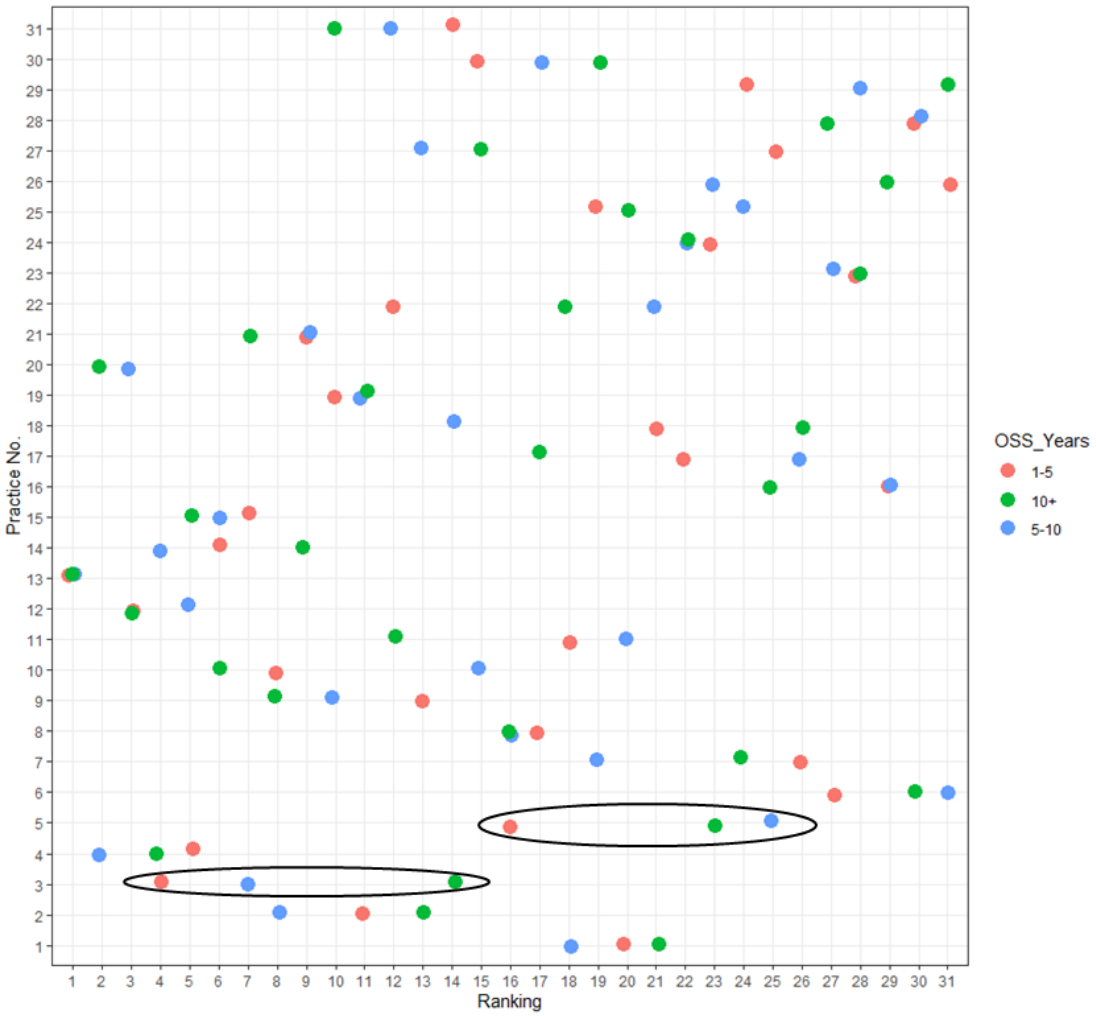

Figure 7.3: Preference trend of contributors on the agreement among three categories of contributors

A visual inspection of 7.3 suggests that there is general agreement among contributors of varying years of experience on OSS projects concerning the effectiveness of practice. And at the extremities, for example P13 (Make project documentation publicly accessible (e.g. public access to archives, mailing list, and to open an issue)) and P6 (Introduce non-restrictive commit access to non-core contributors where appropriate), there would appear to be general agreement across all categories in respect of the ranking. That is, all parties agree that making project documentation publicly available it is very effective, and that introducing non-restrictive commit access is not an effective knowledge retention practice.

There are however some notable inconsistencies across the three categories. For example, in the case of P3 (Use bug labelling so that contributors can effectively select tasks and make contributions (e.g. "suited for newcomers", "Feature xyz")), there is a significant difference in the ranking suggested by those with many years' experience on OSS projects and those with relatively few years of experience. In the case of the latter group, there would appear to be a strong desire to get information about bug so that they can get involved in their resolution. However, in the case of the former group, this is not considered to be quite so important. Perhaps it is the case that newer OSS contributors are eager to find defect resolution work and establish their technical bona fides. For established contributors, this is perhaps much less of a pressing concern and they may already benefit from relationships in the OSS projects that more readily allow them to discover information concerning specific defects.

In the case of P5 (Label substantial new items of work as "work in progress" indicating an opportunity for more contributions), we also see that those with fewer years of OSS project experience seek to have more information about new work items, which might be interpreted as a desire to get more information and to become more established on various OSS projects. More established contributors seem less convinced of the effectiveness of P5, perhaps because they already have a foothold and can target (or maybe even create) new work items through established project channels.

7.3 Evaluating Practice Preference - Number of Years in Computer Programming

Figure 7.4 depicts the ranking of practice effectiveness based on the number of years of programming experience held by participants. As with the evaluation of practice effectiveness based on number of years in OSS projects, a general observation can be made that there is general agreement in the main on the

ranked order of practice effectiveness, irrespective of the number of years of general programming experiences. There are perhaps three notable exceptions to this: P19, P8 and P4.

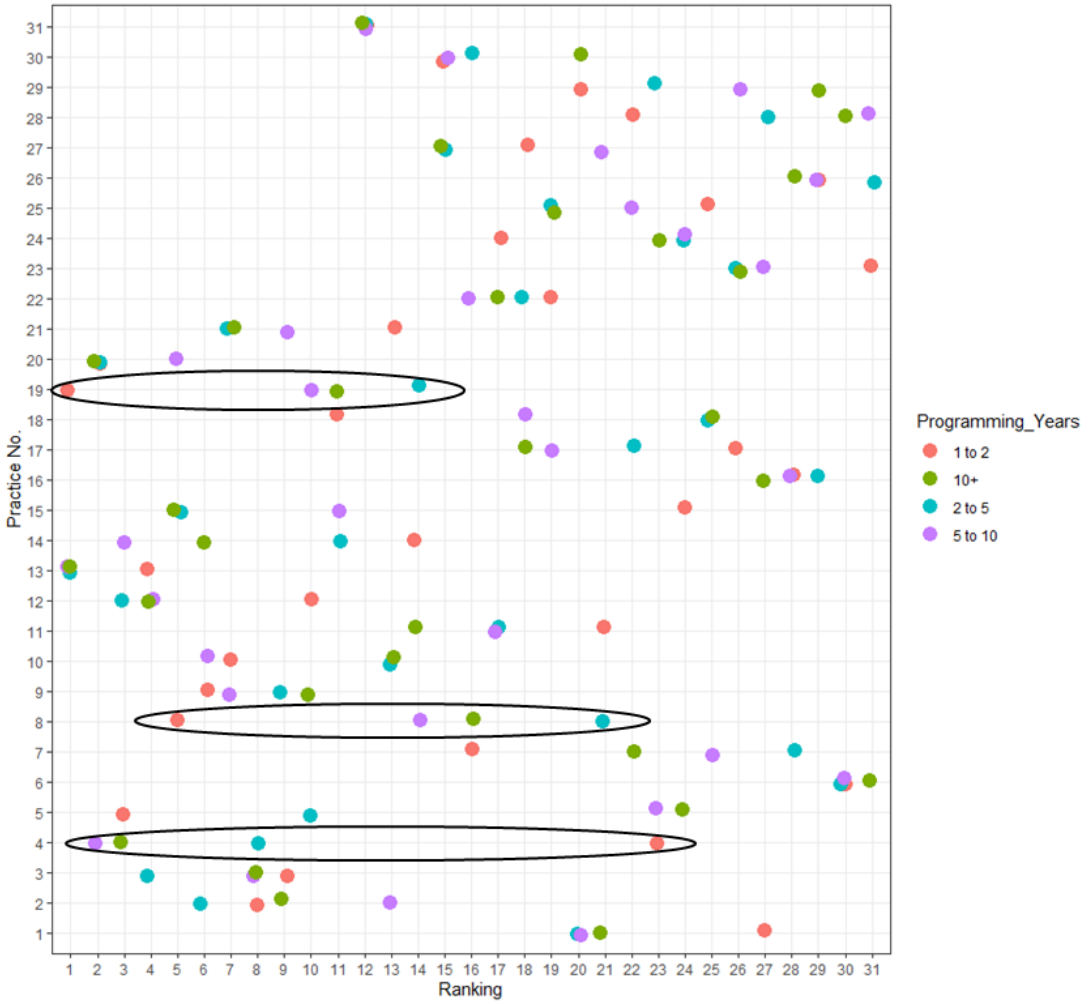

Figure 7.4: Ranking of practices on their effectiveness by contributors with varying experience in programming

In the case of P19 (Encourage open discussion to resolve matters concerning the project community), we see that those with the least years of experience rank open discussion very highly. This may be interpreted as new entrants to the programming field, eager to learn from more experienced members and with a strong desire for open communication to support this objective. In contrast, those with greater levels of general programming experience may not benefit as much from such open discussions. Rather, it might be the case that these

discussions hold only major immediate benefit for inexperienced staff, with those who are more experienced having to give of their time for the benefit of more junior contributors. As in some of the earlier evaluation, here we might again be seeing self-interest and personal value concerns emerging in the data: those who are more experienced have perhaps increased value and power on projects, and training newer arrivals might act to erode this position. Of course, some of this can presumably also be attributed to programmers enjoying programming as opposed to training new programmers.

With P8 (Publish findings of post-mortem reviews such as lesson learned on the project to all contributors), a similar trend again emerges: those newer to the general programming field are keen to get information about OSS projects; in this case, information about lessons learned from post-mortem reviews. This contrasts with more experienced programmers who seem considerably less concerned about obtaining access to post-mortem reviews. While the reasons for this cannot be determined in the scope of this work, the suggested interpretation is that at least in part, the discrepancy in respect of P8 may boil down to the desire of new entrants to the field to get as much information as possible.

In the case of P4 (Ensure the presence of testing artefacts (e.g. unit tests, test scripts, test cases), new entrants to programming have ranked this practice substantially lower than their more experienced counterparts. This may be a simple case of inexperienced programmers having an underappreciation for the importance of testing artefacts, especially on OSS project where contributors are transient and typically not contractually tied. It can furthermore be seen in the case of P4 that the more experienced programmers are, the more importance that is assigned to the presence of testing artefacts. This is perhaps the result of lessons learned through experience: one of the more effective ways to increase confidence in software work products is to attempt to control quality through testing (though clearly this is only one aspect of overall software quality assurance).

7.4 Evaluating Practice Preference - Different Roles in OSS projects

As a general observation, the ranking of OSS knowledge retention practice effectiveness does not seem to vary greatly depending on the contributor role type. In some cases, there is in fact a very high level of clustering to be observed, see for example practices P4, P12, P13, and P20 in 7.5.

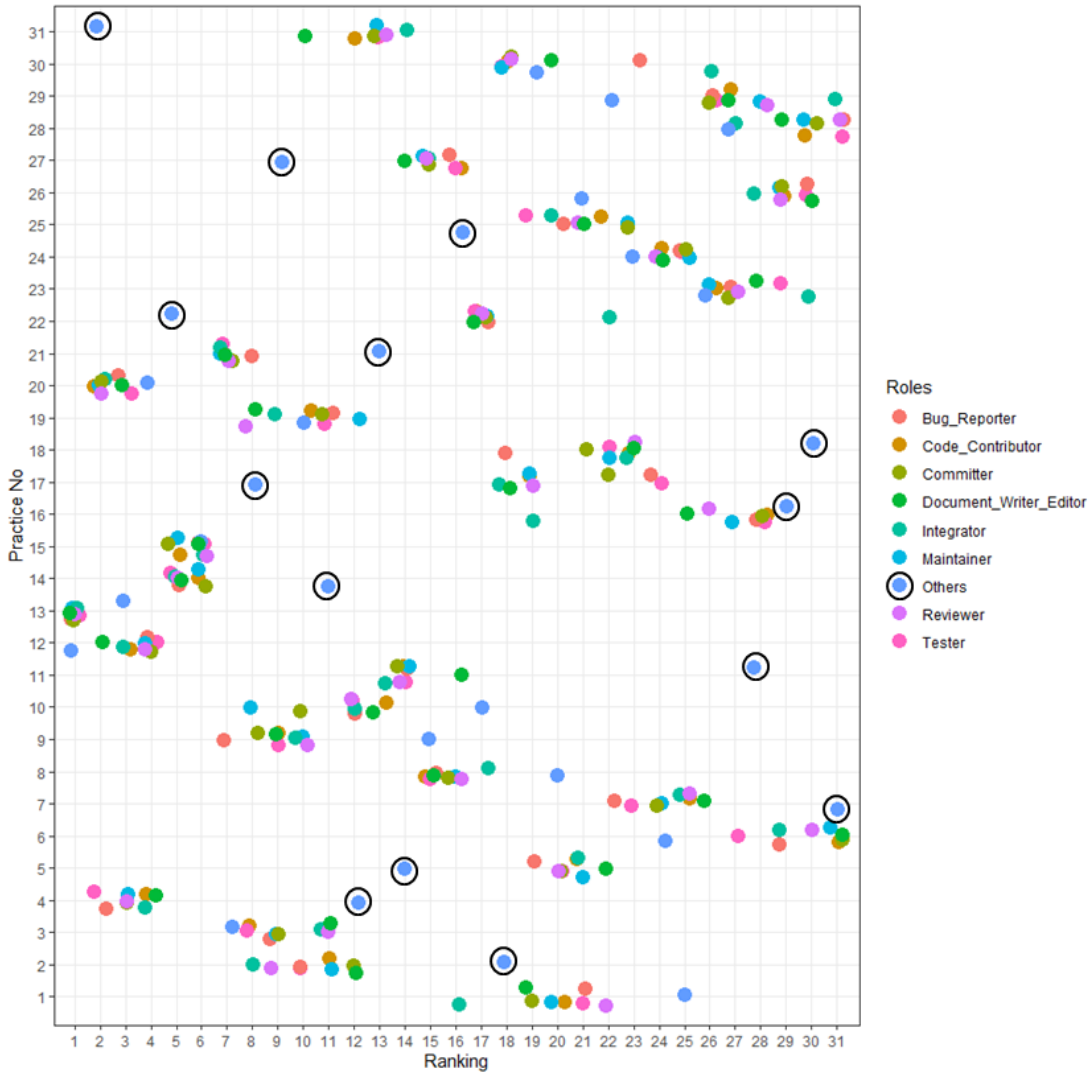

Figure 7.5: The ranking trend of practices by contributors performing tasks in varying roles in OSS projects.

If there is one major further observation that can immediately be seen in 7.5, it is that the role type "others" is frequently presenting as an outlier to otherwise

strongly clustered rankings. The contributors in the role of "others" identify themselves mainly as community managers and as such may not, for example, require access to documentations because they already occupy a central position in the community and are given an access without any hindrance. The preference of community managers is P9, which advocates building a supportive community for conflict resolution. However, since there were only three "other" role type participants across the entire population, the outlying nature of their rankings should be largely overlooked as the number of respondents is too low from which to base any general interpretation.

The role of document editor / writer might be considered to be a marginal outlier in respect of a small number of practices; specifically, P31, P11, P5. In the case of P31 (Promote a culture of appreciation for knowledge contributors (e.g. words of appreciation through blogs and newsletters)), this practice is ranked marginally higher than for the various other contributor types. It might be inferred from this that document writers may feel somewhat underappreciated, that perhaps the more programming-oriented contributions to OSS projects eclipse document writers when it comes to being appreciated (or feeling appreciated).

For P11 (Allow self-organization of project roles) and P5 (Label substantial new items of work as "work in progress" indicating an opportunity for more contributions), the document editor / writer role ranks these only marginally lower than other OSS project role types. Document writers might see self-organisation as less important as it may make their task of writing documents more difficult as it may become more challenging to identify the contributor(s) of interest when documenting certain features. They may also see the opportunity for more technical contributions as promoted in P5 as creating additional documentation work for them personally, and perhaps sometimes these new project additions might not seem warranted to document writers / editors. In any case, there are only marginal differences in evidence in the

specific data, and these interpretations in respect of document writers / editors should be considered as anecdotal or weak in strength.

7.5 Chapter Summary

In terms of the number of OSS projects worked on, the data demonstrates a certain level of homogeneity in terms of reported knowledge retention practice effectiveness. However, there are also some interesting inconsistencies. For example, there is evidence in the data that those on the fringes of OSS projects may desire to move closer to the core, while at the same time, the data suggests that those at the core of OSS projects may not have a strong desire to support newcomers into core positions on projects.

There is also some evidence that relatively inexperienced OSS project contributors and programmers may seek to use OSS projects as a means to improve their knowledge, value and reputation. In the data, we see that relatively inexperienced contributors want quick feedback on code reviews and opportunities for recognition and advancement; more so than their more experienced OSS project colleagues. And perhaps in the context of OSS projects, it is not appropriate, valid or fair to expect experienced and highly specialised enthusiasts and volunteers to start spending less time on activities that they enjoy (i.e. coding and contributing to coding projects) in order to train more inexperienced OSS project members. Maybe there is simply no personal value proposition in such training work for experienced OSS contributors and programmers; and perhaps it is appropriate to expect inexperienced contributors to have to work hard to demonstrate their abilities and worth to a project. However, in cases where OSS projects are large or growing quickly, it is clearly the case that increased attention would need to be focused on on-boarding and this could be achieved by promoting some of the practices that highly experienced contributors seem to undervalue in the data

obtained in this study.

From several difference experience perspectives, there is clear evidence in the data that the presence of testing artefacts is ranked highly, this is especially true for more experienced contributors and programmers who rank this practice considerably higher in some cases. Higher levels of experience on OSS projects (and programming in general), confer more importance on testing artefacts. Where workforces are transient and not contractually bound, it is clearly beneficial that testing artefacts are created as a means to verifying other code based contributions.

As a general statement, the ranking of individual knowledge retention practices is relatively coherent across all OSS contributor types; what variation arises is largely in the context of experience where the inexperienced desire more experience and centrality on OSS projects, while the more experienced OSS campaigners might prefer to spend time doing the things they like best while sustaining their position, reputation and centrality on projects. Or put differently: much of what can be seen in this evaluation can be accounted for by basic human behaviour.

Chapter 8

Conclusion

8.1 Research Overview

This work set out to investigate the following central research hypothesis.

> Knowledge retention practices can be adopted towards effectively addressing knowledge loss in OSS projects.

The research questions investigated in this research are:

> **RQ1.** What is the existing state-of-the-art literature on knowledge loss due to turnover in OSS projects?
>
> **RQ2.** What are the effective knowledge retention practices in OSS projects?

While investigating RQ2, the following two sub-questions emerged, which are iterated as follows:

> **RQ2.1** How can a comprehensive set of knowledge retention practices be developed for OSS projects?
>
> **RQ2.2** How can OSS knowledge retention be evaluated in OSS projects?

To address research question RQ1, a systematic literature review methodology was employed. The snowballing methodology identifies the relevant literature

on knowledge loss due to contributor turnover in OSS projects. Snowballing is considered an efficient and reliable way to conduct a systematic literature review, providing a robust alternative to mechanically searching individual databases for given topics. The robust methodology adopted for the literature review includes a research question, search strategy, inclusion, exclusion, quality criteria, and data synthesis. The snowballing technique is rigorously applied and the outcome of the literature review itself is published separately (Rashid et al., 2019b).

The research methodology adopted in this work is based on mixed methods research and is published in a conference paper (Rashid et al., 2018) and an extended version is published in a journal paper on invitation (Rashid et al., 2019a).

In order to address research question RQ2 in entirety, sub questions RQ 2.1 and RQ 2.2 were investigated. RQ2.1 was addressed by following a systematic process to develop a proactive knowledge retention canonical model in OSS projects. In order to develop the PKR canonical model, candidate practices are selected as data components from disparate sources, which are classified into categories by applying Grounded Theory principles of coding, memoing, and constant comparison to systematically and rigorously distil the set of knowledge retention practices for OSS projects. A total of 31 practices were ultimately identified and classified.

Equipped with the comprehensive set of proactive OSS knowledge retention practices and a robust methodology, the research next moved to answer RQ2.2 by developing a survey instrument that could be employed to evaluate the effectiveness of knowledge retention practices in OSS projects. Over a period of 2 months, a survey instrument was carefully developed, and ultimately prepared for online deployment.

The survey was deployed in the time period of 08-04-2019 to 15-06-2019, after which the data was carefully examined for validity and subjected to a range of analyses and evaluations. A total of 126 valid responses were received

for the survey (which represented a response rate of 1.29%). The profiling of OSS contributors demonstrates that a broad spectrum of OSS contributors have participated in this study. The data sample is representative of various OSS contributors contributing in different roles, with varying levels of experience in OSS and programming ranging from relatively inexperienced to highly experienced. The result of the data analysis is detailed in chapter 6 and the corresponding evaluation of practices is presented in chapter 7 of this thesis.

8.2 Primary Impacts

There are many individual contributions in this work but those of major impact are as follows:

- A comprehensive set of OSS project Knowledge Retention (KR) practices was developed for the first time in this research (presented as KR canonical model. When discharging the survey instrument, respondents were asked to propose any additional practices, but none were forthcoming which demonstrates the comprehensive nature of the KR practices assembled in this work. The effectiveness of the practices was evaluated though the application of a survey instrument in the OSS community, thereby proving the research hypothesis.

- Confirmed that the OSS role types adopted in this work is comprehensive (they could be extended to include the role of Community Manager although these are very few in number).

- A robust survey was utilised to help develop a ranking for OSS KR practice effectiveness. The survey instrument could be further utilised to inquire of the opinion of OSS contributors and to assess and improve the knowledge exchange mechanisms in projects and overall community health.

- The results from the survey demonstrate that there are some clear differences among certain sub-groups of the OSS population, tending to be differentiated largely on the level of experience with OSS and programming in general. There is evidence that relatively inexperienced OSS project contributors and programmers may seek to use OSS projects as a means to improve their knowledge, value and reputation. From the data analysis, it is observable that relatively inexperienced contributors want quick feedback on code reviews and opportunities for recognition and advancement; more so than their more experienced OSS project colleagues.

  In contrast to less experienced contributors, those with more experience and established recognition, report relatively less value in certain practices, for example the establishment of non-core knowledge sharing practices, establishing training mechanisms, and creating knowledge contribution recognition programmes. The explanation for this phenomenon is worthy of further research but may present with a multifaceted set of reasons. On the one hand, contributors with high experience have already less to gain in a social power context from assisting juniors who may ultimately challenge their power. This argument, however, is not likely to be so simple. Perhaps more experienced contributors are in some cases more gifted programmers, and their OSS contribution enjoyment may stem from what are essentially programming related tasks. Furthermore, some junior staff may require very high levels of hand-holding and could seek to adopt the benevolent support of senior contributors only as a basis for their personal training and advancement. Upon detailed examination therefore, the reasons for such a phenomenon could prove complex. Nevertheless, for large and growing OSS projects, there could be significant benefits in adopting the KR practices proposed in this research (as discussed in recommendation 3

below).

8.3 Recommendations for OSS projects

- **Recommendation 1.** In general, use the OSS KR practices developed in this work as a means to understand KR strengths and weaknesses.

- **Recommendation 2.** Apply the general KR ranking schema developed in this work in order to maximise the impact of KR investments in projects. As a general strategy, start by implementing those practices on the higher level of the pyramid in Figure 6.6.

- **Recommendation 3.** If the OSS project is experiencing growth or if the project is large in size, strongly consider promoting some of the practices that have been ranked lowly by experienced OSS campaigners as a means to quickly and efficiently incorporate new contributors to the project. Some examples of practices highly rated by relatively less experienced contributors include P3, P5, P7, and P19. For instance, with P3 (use bug labelling so that contributors can effectively select tasks and make contributions (e.g. "suited for newcomers", "Feature xyz")), contributors with relatively few years of experience, there would appear to be a strong desire to get information about bugs so that they can get involved in their resolution. Similarly, in the case of P5 (Label substantial new items of work as "work in progress" indicating an opportunity for more contributions), contributors with fewer years of OSS project experience seek to have more information about new work items, which might be interpreted as a desire to get more information and to become more established on various OSS projects. Also, P19 (Encourage open discussion to resolve matters concerning the project community), suggests that those with the least years of experience rank open discussion very highly. For certain OSS projects, for example those that are growing

quickly or experiencing relatively higher levels of contributor turnover, the survivability of the project may be enhanced through the adoption of these types of practices.

- **Recommendation 4.** Customise the ranking of practices based on the trend highlighted by relatively experienced and inexperienced contributors in the project. For instance, if an OSS has mostly highly experienced contributors, then practices rated to be highly effective for experienced contributors should be prioritised. On the contrary, for a project with mostly inexperienced contributors the practices identified to be highly effective in this situation should be prioritised. The canonical model classifies practices based on categories including communications, core development practices, governance and leadership, environment/ ecosystem/ culture and contributor motivation. The OSS projects striving to strengthen the KR mechanisms by one focus area should identify their requirements and prioritise practices from that particular area in their work settings. Furthermore, the priority of practices can be changed according to the requirement of the OSS project.

- **Recommendation 5.** The KR practices should be part of the fundamental documents in OSS projects, for example included under project guidelines, manuals of on-boarding minimising knowledge sharing barriers for newcomers (Steinmacher et al., 2015b), and respective manuals on becoming a mentor on the project. This will enable awareness on knowledge retention in OSS communities. Adopting KR practices for Knowledge Management, as a manifesto in OSS projects is highly recommended.

8.4 Research Limitations

While every effort was taken at every stage to guard academic robustness in this research, it does nevertheless exhibit some limitations that should be articulated. In this section limitation concerning the literature review, survey, and overall analysis are discussed.

In the literature review, a SB approach is used to identify relevant papers. It is arguable that there is a possibility that the SB method is weak and does not provide enough coverage of the relevant literature in this review. The SB method is shown in literature to provide coverage similar to database searches (Wohlin, 2014). The SB search strategy can be effectively employed in place of the database search to find relevant papers. In order to overcome the first threat, the rigorous SB steps presented in section 2.3 were adopted.

A second possible threat is the misinterpretation of the concepts during data synthesis, in terms of generalization to all OSS projects in the papers. To overcome the second threat, the dynamic nature of OSS projects is considered, which can be purely volunteer based or hybrid with commercial involvements. Not every OSS project community follow the same policies and practices and they vary constantly. The third threat in this literature review is that the concepts consolidated in this review are from the common understanding of the researcher, therefore themes in this work may be subjective and personal to some extent.

There is a possibility that some relevant literature was missed while reading the paper titles in BSB and FSB. In the case where the text of the title was not clear, abstracts and conclusions of the papers were examined to ensure that important information was not disregarded in the search. The objective of this review was to include papers that discussed KL in OSS projects due to contributor turnover. Papers that use OSS to design technological solutions for knowledge storage were not included in this literature review.

In relation to the survey and representation of OSS contributors, the limitations are as follows:

- In future work, the limit of extracting participants from 1020 projects on GitHub can be extended.

- This work does not account for cases where participants have more than one account existing on GitHub with different e-mails addresses. There is a possibility that they each receive a copy of the survey. This work assumes that each participant only took the survey once.

- The survey participants were genuinely engaged with the research theme and appeared ethically sound regarding their input. The responses collected through Google Forms were inspected individually for relevant validity details such as the removal of certain responses are discussed in chapter 5.

- This work is limited in that of the survey respondents, just 2% reported having less than 2 years programming experience. This is considered to be a limitation because one aspect of this work seeks to develop a knowledge retention practice reference list for use by all contributor types, and perhaps importantly, to elicit the views of those who are relatively inexperienced and who seek to be involved in OSS projects. However, upon more detailed consideration, it may be the case that these are in fact student programmers or those learning to program and that they are not yet at the stage where they might be able to contribute to OSS projects. Since the survey did not request an explanation where respondents have less than 2 years of programming experience, it is not possible to identify a firm conclusion in this respect.

  In terms of data analysis the limitations are as follows:

- The classification of practices into groups is based on the absolute mean

value for the effectiveness of each practice. This is a somewhat crude grouping mechanism but having examined various other grouping techniques it proved to be the most understandable across all the various analyses. It is nevertheless accepted that this is an unsophisticated and somewhat arbitrary grouping mechanism and that it allows for a practice with an mean effectiveness of 6.999 to be grouped separately to a practice with a mean effectiveness of 7.001, when in fact there is very little difference between these two mean values. In this sense, the grouping could be misinterpreted and therefore it is important to highlight this limitation. The benefit of the chosen grouping technique is that it allows for a single interpretation of groupings across the various categories of analysis.

- The focus of this research does not consider different OSS project licensing schemes, and this may have an impact on the flow of knowledge among contributors due to constraints imposed by the commercial involvement of the project. This could form part of the scope of extended future research.

8.5 Future Work

1. Perform qualitative work relevant to KR mechanisms and study the relationship among the practices proposed and evaluated in this work.

2. Examine OSS projects of different sizes and profiles to fully articulate the specific needs of specific OSS projects in terms of knowledge retention practices.

3. Develop an OSS KR practice assessment tool that projects can use to rate their own KR level. This could be a standard devised based on the KR canonical model used for the evaluation of OSS projects, determining which practices are to be implemented or improved. The KR practices can be

operationalised by incorporating mechanisms that motivate contributors at an intrinsic level to follow them.

4. Examine the possibility of adopting KR practices in OSS projects in agile development environments, and perhaps in general software engineering also, especially globally distributed.

Bibliography

Adams, P. J., Capiluppi, A., and Boldyreff, C. (2009). Coordination and productivity issues in free software: The role of brooks' law. In *2009 IEEE International Conference on Software Maintenance*, pages 319–328. IEEE.

Aggestam, L., Söderström, E., and Persson, A. (2010). Seven types of knowledge loss in the knowledge capture process. In *ECIS*, page 13.

Allan, G. (2003). A critique of using grounded theory as a research method. *Electronic journal of business research methods*, 2(1):1–10.

Allen, I. E. and Seaman, C. A. (2007). Likert scales and data analyses. *Quality progress*, 40(7):64–65.

Anquetil, N., de Oliveira, K. M., de Sousa, K. D., and Dias, M. G. B. (2007). Software maintenance seen as a knowledge management issue. *Information and Software Technology*, 49(5):515–529.

Badampudi, D., Wohlin, C., and Petersen, K. (2015). Experiences from using snowballing and database searches in systematic literature studies. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, page 17. ACM.

Baheti, P., Gehringer, E., and Stotts, D. (2002). Exploring the efficacy of distributed pair programming. In *Conference on Extreme Programming and Agile Methods*, pages 208–220. Springer.

Bao, L., Xing, Z., Xia, X., Lo, D., and Li, S. (2017). Who will leave the company?: a large-scale industry study of developer turnover by mining monthly work

report. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 170–181. IEEE.

Basili, V. R. (1990). Viewing maintenance as reuse-oriented software development. *IEEE software*, 7(1):19–25.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Manifesto for agile software development.

Boone, H. N. and Boone, D. A. (2012). Analyzing likert data. *Journal of extension*, 50(2):1–5.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of github repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344. IEEE.

Bosu, A. and Carver, J. C. (2014). Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*, page 33. ACM.

Britto, R., Smite, D., and Damm, L.-O. (2016). Software architects in large-scale distributed projects: An ericsson case study. *IEEE Software*, 33(6):48–55.

Bryant, A. and Charmaz, K. (2007). *The Sage handbook of grounded theory*. Sage.

Capiluppi, A., Gonzalez Barahona, J. M., and Herraiz, I. (2007). Adapting the "staged model for software evolution" to floss. In *to FLOSS', ESEC/FSE'07 Joint 11th European Software Engineering Conference (ESEC) and 15th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-13) 2007, Cavat near Dubrovnik, Croatia, 03-07 September*.

Capiluppi, A. and Michlmayr, M. (2007). From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects. In *IFIP International Conference on Open Source Systems*, pages 31–44. Springer.

Capiluppi, A., Stol, K.-J., and Boldyreff, C. (2012). Exploring the role of commercial stakeholders in open source software evolution. In *IFIP International Conference on Open Source Systems*, pages 178–200. Springer.

Charmaz, K. (2006). *Constructing grounded theory: A practical guide through qualitative analysis*. sage.

Chen, X., Li, X., Clark, J. G., and Dietrich, G. B. (2013). Knowledge sharing in open source software project teams: A transactive memory system perspective. *International Journal of Information Management*, 33(3):553–563.

Choi, B., Poon, S. K., and Davis, J. G. (2008). Effects of knowledge management strategy on organizational performance: A complementarity theory-based approach. *Omega*, 36(2):235–251.

Ciborra, C. U. and Andreu, R. (2001). Sharing knowledge across boundaries. *Journal of Information technology*, 16(2):73–81.

Ciolkowski, M., Laitenberger, O., Vegas, S., and Biffl, S. (2003). Practical experiences in the design and conduct of surveys in empirical software engineering. In *Empirical methods and studies in software engineering*, pages 104–128. Springer.

Clarke, P., O'Connor, R. V., and Leavy, B. (2016). A complexity theory viewpoint on the software development process and situational context. In *Proceedings of the International Conference on Software and Systems Process*, pages 86–90. ACM.

Clarke, P. and O'Connor, R. V. (2012). The situational factors that affect the

software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447.

Clason, D. L. and Dormody, T. J. (1994). Analyzing data measured by individual likert-type items. *Journal of agricultural education*, 35(4):4.

Coelho, J. and Valente, M. T. (2017). Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 186–196. ACM.

Colazo, J. and Fang, Y. (2009). Impact of license choice on open source software development activity. *Journal of the American Society for Information Science and Technology*, 60(5):997–1011.

Collier, B., DeMarco, T., and Fearey, P. (1996). A defined process for project post mortem review. *IEEE software*, 13(4):65–72.

Constantinou, E. and Mens, T. (2017a). An empirical comparison of developer retention in the rubygems and npm software ecosystems. *Innovations in Systems and Software Engineering*, 13(2-3):101–115.

Constantinou, E. and Mens, T. (2017b). Socio-technical evolution of the ruby ecosystem in github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 34–44. IEEE.

Corbin, J. and Strauss, A. (2008). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. sage.

Corbin, J. M. and Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1):3–21.

Costa, E., Soares, A. L., and de Sousa, J. P. (2016). Information, knowledge and collaboration management in the internationalisation of smes: a systematic literature review. *International Journal of Information Management*, 36(4):557–569.

Costantino, T. E. (2001). Narrative inquiry: Experience and story in qualitative research.

Creswel, J. W. (2009). Research design: Qualitative, quantitative, and mixed methods approaches. *Los angeles: University of Nebraska–Lincoln.*

Creswell, J. (2014). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications.

Creswell, J. and Clark, V. (2011). *Designing and Conducting Mixed Methods Research*. SAGE Publications.

Crowston, K. (2011). Lessons from volunteering and free/libre open source software development for the future of work. In *Researching the Future in Information Systems*, pages 215–229. Springer.

Crowston, K., Annabi, H., Howison, J., and Masango, C. (2004). Effective work practices for software engineering: free/libre open source software development. In *Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research*, pages 18–26. ACM.

Crowston, K. and Howison, J. (2005). The social structure of free and open source software development. *First Monday*, 10(2).

Crowston, K., Howison, J., and Annabi, H. (2006). Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice*, 11(2):123–148.

Crowston, K., Wei, K., Howison, J., and Wiggins, A. (2012). Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):7.

Curtin, R., Presser, S., and Singer, E. (2000). The effects of response rate changes on the index of consumer sentiment. *Public opinion quarterly*, 64(4):413–428.

Cusumano, M. A. and Selby, R. W. (1998). *Microsoft secrets: how the world's most powerful software company creates technology, shapes markets, and manages people*. Simon and Schuster.

Dafermos, G. N. (2005). Management and virtual decentralised networks: The linux project (originally published in volume 6, number 11, november 2001). *First Monday*.

Daghfous, A., Belkhodja, O., and C. Angell, L. (2013). Understanding and managing knowledge loss. *Journal of Knowledge Management*, 17(5):639–660.

Davenport, T. H., Prusak, L., et al. (1998). *Working knowledge: How organizations manage what they know*. Harvard Business Press.

De Long, D. W. and Davenport, T. (2003). Better practices for retaining organizational knowledge: Lessons from the leading edge. *Employment Relations Today*, 30(3):51.

DeBrie, E. and Goeschel, D. (2016). Open source software licenses.

DeLong, T. J. (1982). Reexamining the career anchor model. *Personnel*, 59(3):50–61.

Denzin, N. K. (1973). *The research act: A theoretical introduction to sociological methods*. Transaction.

Denzin, N. K. and Lincoln, Y. S. (2011). *The Sage handbook of qualitative research*. Sage.

Dingsøyr, T. (2005). Postmortem reviews: purpose and approaches in software engineering. *Information and Software Technology*, 47(5):293–303.

Dingsøyr, T., Bjørnson, F. O., and Shull, F. (2009). What do we know about knowledge management? practical implications for software engineering. *IEEE software*, 26(3):100–103.

Dingsøyr, T. and Conradi, R. (2002). A survey of case studies of the use of knowledge management in software engineering. *International journal of software engineering and knowledge engineering*, 12(04):391–414.

Dinh-Trong, T. and Bieman, J. M. (2004). Open source software development: a case study of freebsd. In *10th International Symposium on Software Metrics, 2004. Proceedings.*, pages 96–105. IEEE.

Doan, Q. M., Rosenthal-Sabroux, C., and Grundstein, M. (2011). A reference model for knowledge retention within small and medium-sized enterprises. In *KMIS*, pages 306–311.

Donadelli, S. M. (2015). *The impact of knowledge loss on software projects: turnover, customer found defects, and dormant files*. PhD thesis, Concordia University.

Donnellan, B., Fitzgerald, B., Lake, B., and Sturdy, J. (2005). Implementing an open source knowledge base. *IEEE Software*, 22(6):92–95.

Douglas, D. (2003). Grounded theories of management: A methodological review. *Management Research News*, 26(5):44–52.

Droege, S. B. and Hoobler, J. M. (2003). Employee turnover and tacit knowledge diffusion: A network perspective. *Journal of Managerial Issues*, pages 50–64.

Drucker, P. F. (1999). Knowledge-worker productivity: The biggest challenge. *California management review*, 41(2):79–94.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9-10):833–859.

Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. (2008). Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer.

Eghbal, N., Keepers, and B., Wills, S. (2018). Open source guides. URL:`https://opensource.guide/`, (visited on 01/15/2019).

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of management review*, 14(4):532–550.

Emanuel, A. W. R. (2014). Statistical analysis of popular open source software projects and their communities. In *2014 6th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–6. IEEE.

Fægri, T. E., Dybå, T., and Dingsøyr, T. (2010). Introducing knowledge redundancy practice in software development: Experiences with job rotation in support work. *Information and Software Technology*, 52(10):1118–1132.

Felizardo, K. R., Mendes, E., Kalinowski, M., Souza, É. F., and Vijaykumar, N. L. (2016). Using forward snowballing to update systematic reviews in software engineering. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 53. ACM.

Feller, J., Fitzgerald, B., et al. (2002). *Understanding open source software development*. Addison-Wesley London.

Fink, A. (2003). *The survey handbook*. Sage.

Fitzgerald, B. (2006). The transformation of open source software. *MIS quarterly*, pages 587–598.

Flick, U. (2018). *Doing Grounded Theory*. Qualitative Research Kit. SAGE Publications.

Foucault, M., Palyart, M., Blanc, X., Murphy, G. C., and Falleri, J.-R. (2015). Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 829–841. ACM.

Fronza, I., Janes, A., Sillitti, A., Succi, G., and Trebeschi, S. (2013). Cooperation wordle using pre-attentive processing techniques. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 57–64. IEEE.

Gammelgaard, J. (2007). Why not use incentives to encourage knowledge sharing. *Journal of Knowledge Management Practice*, 8(1):115–123.

Garcia, D., Zanetti, M. S., and Schweitzer, F. (2013). The role of emotions in contributors activity: A case study on the gentoo community. In *2013 International Conference on Cloud and Green Computing*, pages 410–417. IEEE.

Ge, X., Dong, Y., and Huang, K. (2006). Shared knowledge construction process in an open-source software development community: An investigation of the gallery community. In *Proceedings of the 7th international conference on Learning sciences*, pages 189–195. International Society of the Learning Sciences.

Geiger, R. S. (2017). Summary analysis of the 2017 github open source survey. *arXiv preprint arXiv:1706.02777*.

Ghiselli, E., Campbell, J., and Zedeck, S. (1981). *Measurement Theory for the Behavioral Sciences*. A Series of books in psychology. W. H. Freeman.

Giorgi, A. (2009). *The descriptive phenomenological method in psychology: A modified Husserlian approach.* Duquesne University Press.

Glaser, B. (1992). *Emergence Vs Forcing: Basics of Grounded Theory Analysis*. Sociology Press.

Goeminne, M. and Mens, T. (2011). Evidence for the pareto principle in open source software activity. In *the Joint Porceedings of the 1st International*

workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability, pages 74–82. Citeseer.

Gousios, G., Zaidman, A., Storey, M.-A., and Van Deursen, A. (2015). Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 358–368. IEEE Press.

Hagan, D., Watson, O., and Barron, K. (2007). Ascending into order: A reflective analysis from a small open source development team. *International Journal of Information Management*, 27(6):397–405.

Hansen, M. T., Nohria, N., and Tierney, T. (1999). What's your strategy for managing knowledge. *The knowledge management yearbook 2000–2001*, 77(2):106–116.

Hemetsberger, A. and Reinhardt, C. (2004). Sharing and creating knowledge in open-source communities: the case of kde. In *Paper for Fifth European Conference on Organizational Knowledge, Learning, and Capabilities, Innsbruck*.

Herbsleb, J. D. and Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on software engineering*, 29(6):481–494.

Herraiz, I., Robles, G., Amor, J. J., Romera, T., and González Barahona, J. M. (2006). The processes of joining in global distributed software projects. In *Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 27–33. ACM.

Hoffart, N. (2000). Basics of qualitiative research: Techniques and procedures for developing grounded theory. *Nephrology Nursing Journal*, 27(2):248.

Holton, J. A. (2007). The coding process and its challenges. *The Sage handbook of grounded theory*, pages 265–89.

Hsu, C.-L. and Lin, J. C.-C. (2008). Acceptance of blog usage: The roles of technology acceptance, social influence and knowledge sharing motivation. *Information & management*, 45(1):65–74.

Huang, P. and Zhang, Z. (2013). Participation in open knowledge communities and job-hopping: evidence from enterprise software. *MIS Quarterly, Forthcoming*.

Huber, G. P. (1996). Organizational learning: a guide for executives in technology–critical organizations. *International Journal of Technology Management*, 11(7-8):821–832.

Hutchison, C. S. (2001). Personal knowledge, team knowledge, real knowledge. In *EUROCON'2001. International Conference on Trends in Communications. Technical Program, Proceedings (Cat. No. 01EX439)*, volume 1, pages 247–250. IEEE.

Huysman, M. and Lin, Y. (2005). Learn to solve problems: A virtual ethnographic case study of learning in s gnu/linux users group. *The Electronic Journal for Virtual Organizations and Networks*, 7.

Huysman, M. H., Lin, Y., et al. (2006). Learn to solve problems: a virtual ethnographic case study of learning in a gnu/linux users group.

Izquierdo, J. L. C., Cosentino, V., and Cabot, J. (2017). An empirical study on the maturity of the eclipse modeling ecosystem. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 292–302. IEEE.

Izquierdo-Cortazar, D., Robles, G., Ortega, F., and Gonzalez-Barahona, J. M. (2009). Using software archaeology to measure knowledge loss in software

projects due to developer turnover. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–10. IEEE.

Jansen, A. G. J. (2008). *Architectural design decisions*. University Library of Groningen.

Jennex, M. E. and Durcikova, A. (2013). Assessing knowledge loss risk. In *2013 46th Hawaii International Conference on System Sciences*, pages 3478–3487. IEEE.

Jensen, C. and Scacchi, W. (2005). Modeling recruitment and role migration processes in ossd projects. *ProSim05*, 39.

Joblin, M., Apel, S., and Mauerer, W. (2017). Evolutionary trends of developer coordination: A network approach. *Empirical Software Engineering*, 22(4):2050–2094.

Johnson, R. B. and Onwuegbuzie, A. J. (2004). Mixed methods research: A research paradigm whose time has come. *Educational researcher*, 33(7):14–26.

Johnson, R. B., Onwuegbuzie, A. J., and Turner, L. A. (2007). Toward a definition of mixed methods research. *Journal of mixed methods research*, 1(2):112–133.

Keele, S. et al. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

Keeter, S., Kennedy, C., Dimock, M., Best, J., and Craighill, P. (2006). Gauging the impact of growing nonresponse on estimates from a national rdd telephone survey. *International Journal of Public Opinion Quarterly*, 70(5):759–779.

Khondhu, J., Capiluppi, A., and Stol, K.-J. (2013). Is it all lost? a study of inactive open source projects. In *IFIP International Conference on Open Source Systems*, pages 61–79. Springer.

Kitchenham, B. and Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information and software technology*, 55(12):2049–2075.

Kitchenham, B., Pickard, L., and Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE software*, 12(4):52–62.

Kitchenham, B. A. and Pfleeger, S. L. (2008). Personal opinion surveys. In *Guide to advanced empirical software engineering*, pages 63–92. Springer.

Klein, H. K. and Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS quarterly*, 23(1):67–94.

Koh, J. and Kim, Y.-G. (2004). Knowledge sharing in virtual communities: an e-business perspective. *Expert systems with applications*, 26(2):155–166.

Koziolek, H. (2008). Goal, question, metric. In *Dependability metrics*, pages 39–42. Springer.

Kuk, G. (2006). Strategic interaction and knowledge sharing in the kde developer mailing list. *Management science*, 52(7):1031–1042.

Lakhani, K. R. and Von Hippel, E. (2004). How open source software works:“free” user-to-user assistance. In *Produktentwicklung mit virtuellen Communities*, pages 303–339. Springer.

Lakhani, K. R., Von Hippel, E., et al. (2003). How open source software works:" free" user-to-user assistance. *Research Policy*, 32(6):923–943.

Lam, A. (2000). Tacit knowledge, organizational learning and societal institutions: An integrated framework. *Organization studies*, 21(3):487–513.

Lee, G. K. and Cole, R. E. (2003). From a firm-based to a community-based model of knowledge creation: The case of the linux kernel development. *Organization science*, 14(6):633–649.

Lee, S., Baek, H., and Jahng, J. (2017). Governance strategies for open collaboration: Focusing on resource allocation in open source software development organizations. *International Journal of Information Management*, 37(5):431–437.

Leibowitz, J. (2011). Knowledge retention: What practitioners need to know.

Lempert, L. B. (2007). Asking questions of the data: Memo writing in the grounded. *The Sage handbook of grounded theory*, pages 245–264.

Lethbridge, T. C., Sim, S. E., and Singer, J. (2005). Studying software engineers: Data collection techniques for software field studies. *Empirical software engineering*, 10(3):311–341.

Levy, M. and Hazzan, O. (2009). Knowledge management in practice: The case of agile software development. In *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pages 60–65. IEEE.

Licorish, S. A. and MacDonell, S. G. (2014). Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. *Information and Software Technology*, 56(12):1578–1596.

Liebowitz, J. (2008). *Knowledge retention: strategies and solutions*. Auerbach Publications.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

Lin, B., Robles, G., and Serebrenik, A. (2017). Developer turnover in global, industrial open source projects: Insights from applying survival analysis. In *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*, pages 66–75. IEEE.

Lincoln, Y. S., Lynham, S. A., and Guba, E. G. (2011). Paradigmatic controversies, contradictions, and emerging confluences, revisited. *The Sage handbook of qualitative research*, 4:97–128.

Lindvall, M. and Rus, I. (2003). Knowledge management for software organizations. In *Managing software engineering knowledge*, pages 73–94. Springer.

Ling, K., Beenen, G., Ludford, P., Wang, X., Chang, K., Li, X., Cosley, D., Frankowski, D., Terveen, L., Rashid, A. M., et al. (2005). Using social psychology to motivate contributions to online communities. *Journal of Computer-Mediated Communication*, 10(4).

Liu, W., Chen, C. L., Lakshminarayanan, V., and Perry, D. E. (2005). A design for evidence-based soft research. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–7. ACM.

Lungu, M. (2008). Towards reverse engineering software ecosystems. In *2008 IEEE International Conference on Software Maintenance*, pages 428–431. IEEE.

Mackenzie, N. and Knipe, S. (2006). Research dilemmas: Paradigms, methods and methodology. *Issues in educational research*, 16(2):193–205.

Marks, M. A., Mathieu, J. E., and Zaccaro, S. J. (2001). A temporally based framework and taxonomy of team processes. *Academy of management review*, 26(3):356–376.

Menon, T. and Pfeffer, J. (2003). Valuing internal vs. external knowledge: Explaining the preference for outsiders. *Management Science*, 49(4):497–513.

Mens, T. (2016). An ecosystemic and socio-technical view on software maintenance and evolution. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–8. IEEE.

Micciancio, D. and Voulgaris, P. (2011). Knowledge retention: What practitioners need to know. *KM World*, 20(2).

Michlmayr, M. (2007a). Quality improvement in volunteer free and open source software projects. *Opensource. MIT*.

Michlmayr, M. (2007b). *Quality improvement in volunteer free and open source software projects: exploring the impact of release management*. PhD thesis, University of Cambridge.

Mockus, A. (2010). Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 117–126. ACM.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346.

Monus, A. (2017). Remote pair programming: How to make it work for you. `URL:https://dzone.com/articles/remote-, pair-programming-how-to-make-it-work-for-you` (visited on 06/02/2019).

Monus, A. (2018). Remote pair programming: Tips, tools, and how to measure. `URL:https://raygun.com/blog/remote-pair-programming/` (visited on 09/02/2019).

Morton, S. M., Bandara, D. K., Robinson, E. M., and Carr, P. E. A. (2012). In the 21st century, what is an acceptable response rate? *Australian and New Zealand journal of public health*, 36(2):106–108.

Nassif, M. and Robillard, M. P. (2017). Revisiting turnover-induced knowledge loss in software projects. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 261–272. IEEE.

Nidhra, S., Yanamadala, M., Afzal, W., and Torkar, R. (2013). Knowledge transfer challenges and mitigation strategies in global software development—a systematic literature review and industrial validation. *International journal of information management*, 33(2):333–355.

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization science*, 5(1):14–37.

Nonaka, I. and Takeuchi, H. (1995). *The knowledge-creating company: How Japanese companies create the dynamics of innovation*. Oxford university press.

Nonaka, I., Toyama, R., and Konno, N. (2000). Seci, ba and leadership: a unified model of dynamic knowledge creation. *Long range planning*, 33(1):5–34.

Oates, B. J. (2006). *Researching Information Systems and Computing*. SAGE.

Otte, T., Moreton, R., and Knoell, H. D. (2008). Applied quality assurance methods under the open source development model. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 1247–1252. IEEE.

Pee, L. G. and Lee, J. (2015). Intrinsically motivating employees' online knowledge sharing: Understanding the effects of job design. *International Journal of Information Management*, 35(6):679–690.

Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology*, 19(3):295–341.

Petrie, A. and Sabin, C. (2019). *Medical statistics at a glance*. John Wiley & Sons.

Petticrew, M. and Roberts, H. (2008). *Systematic reviews in the social sciences: A practical guide*. John Wiley & Sons.

Pontika, N., Knoth, P., Cancellieri, M., and Pearce, S. (2015). Fostering open science to research using a taxonomy and an elearning portal. In *Proceedings of the 15th international conference on knowledge technologies and data-driven business*, page 11. ACM.

Qin, X., Salter-Townshend, M., and Cunningham, P. (2014). Exploring the relationship between membership turnover and productivity in online communities. In *Eighth International AAAI Conference on Weblogs and Social Media*.

Qumer, A. and Henderson-Sellers, B. (2008). A framework to support the evaluation, adoption and improvement of agile methods in practice. *Journal of Systems and Software*, 81(11):1899–1919.

Ramachandran, K. and Tsokos, C. (2009). *Mathematical Statistics with Applications*. Academic Press.

Ransbotham, S. and Kane, G. C. (2011). Membership turnover and collaboration success in online communities: Explaining rises and falls from grace in wikipedia. *Mis Quarterly*, pages 613–627.

Rashid, M., Clarke, P. M., and O'Connor, R. V. (2019a). A mechanism to explore proactive knowledge retention in open source software communities. *Journal of Software: Evolution and Process*, pages 636–644.

Rashid, M., Clarke, P. M., and O'Connor, R. V. (2017). Exploring knowledge loss in open source software (oss) projects. In *International conference on software process improvement and capability determination*, pages 481–495. Springer.

Rashid, M., Clarke, P. M., and O'Connor, R. V. (2018). An approach to investigating proactive knowledge retention in oss communities. In *European Conference on Software Process Improvement*, pages 108–119. Springer.

Rashid, M., Clarke, P. M., and O'Connor, R. V. (2019b). A systematic examination of knowledge loss in open source software projects. *International Journal of Information Management*, 46:104–123.

Rastogi, A. and Sureka, A. (2014). What community contribution pattern says about stability of software project? In *2014 21st Asia-Pacific Software Engineering Conference*, volume 2, pages 31–34. IEEE.

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49.

Rhem, A. J. (2005). *UML for developing knowledge management systems*. Auerbach Publications.

Riessman, C. K. (2008). *Narrative methods for the human sciences*. Sage.

Rigby, P. C., German, D. M., Cowen, L., and Storey, M.-A. (2014). Peer review on open-source software projects: Parameters, statistical models, and theory. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):35.

Rigby, P. C., Zhu, Y. C., Donadelli, S. M., and Mockus, A. (2016). Quantifying and mitigating turnover-induced knowledge loss: Case studies of chrome and a project at avaya. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 1006–1016. IEEE.

Robles, G. and Gonzalez-Barahona, J. M. (2006). Contributor turnover in libre software projects. In *IFIP International Conference on Open Source Systems*, pages 273–286. Springer.

Robles, G., Gonzalez-Barahona, J. M., and Michlmayr, M. (2005). Evolution of volunteer participation in libre software projects: evidence from debian. In *Proceedings of the 1st international conference on open source systems*, pages 100–107.

Ross, S. M. (2014). *Introduction to probability and statistics for engineers and scientists*. Academic Press.

Rossman, G. B. and Wilson, B. L. (1985). Numbers and words: Combining quantitative and qualitative methods in a single large-scale evaluation study. *Evaluation review*, 9(5):627–643.

Rubin, H. J. and Rubin, I. S. (2011). *Qualitative interviewing: The art of hearing data*. Sage.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131.

Rus, I., Lindvall, M., and Sinha, S. (2002). Knowledge management in software engineering. *IEEE software*, 19(3):26–38.

Ryan, S. and O'Connor, R. V. (2013). Acquiring and sharing tacit knowledge in software development teams: An empirical study. *Information and Software Technology*, 55(9):1614–1624.

Scacchi, W. (2003). Free/open source software development practices in the computer game community. *Institute for Software Research, University of California, Technical Report*.

Scacchi, W. (2007). Free/open source software development: recent research results and emerging opportunities. In *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers*, pages 459–468. ACM.

Schilling, A., Laumer, S., and Weitzel, T. (2011). Is the source strong with you? a fit perspective to predict sustained participation of floss developers.

Schilling, A., Laumer, S., and Weitzel, T. (2012). Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss

projects. In *2012 45th Hawaii International Conference on System Sciences*, pages 3446–3455. IEEE.

Schubert, P., Lincke, D.-M., and Schmid, B. (1998). A global knowledge medium as a virtual community: the netacademy concept. *AMCIS 1998 Proceedings*, page 207.

Shah, S. K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management science*, 52(7):1000–1014.

Sharif, K. Y., English, M., Ali, N., Exton, C., Collins, J., and Buckley, J. (2015). An empirically-based characterization and quantification of information seeking through mailing lists during open source developers' software evolution. *Information and Software Technology*, 57:77–94.

Shull, F., Singer, J., and Sjøberg, D. I. (2008). Guide to advanced empirical software engineering.

Silic, M. and Back, A. (2017). Open source software adoption: lessons from linux in munich. *IT Professional*, 19(1):42–47.

Singh, V., Twidale, M. B., and Rathi, D. (2006). Open source technical support: A look at peer help-giving. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 6, pages 118c–118c. IEEE.

Software, B. D. (2015). It's an open-source world: 78 percent of companies run open-source software. URL: `https://www.zdnet.com/article/its-an-open-source-world-78-percent-of-companies-run-open-source-software.html`, (visited on 08/06/2017).

Somekh, B. and Lewin, C. (2005). *Research Methods in the Social Sciences*. SAGE Publications.

Sowe, S. K., Karoulis, A., and Stamelos, I. (2006). A constructivist view of knowledge management in open source virtual communities. In *Managing learning in virtual settings: the role of context*, pages 290–308. IGI Global.

Sowe, S. K., Stamelos, I., and Angelis, L. (2008). Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software*, 81(3):431–446.

Steinmacher, I., Conte, T., Gerosa, M. A., and Redmiles, D. (2015a). Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*, pages 1379–1392. ACM.

Steinmacher, I., Silva, M. A. G., Gerosa, M. A., and Redmiles, D. F. (2015b). A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67–85.

Strauss, A. and Corbin, J. M. (1998). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. SAGE Publications.

Subramaniam, C., Sen, R., and Nelson, M. L. (2009). Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, 46(2):576–585.

Sundaram, K., Dwivedi, S., and Sreenivas, V. (2010). *Medical Statistics: Principles & Methods*. Anshan.

Tashakkori, A. and Teddlie, C. (2010). *Sage handbook of mixed methods in social & behavioral research*. sage.

Tiwana, A. (2004). An empirical study of the effect of knowledge integration on software development performance. *Information and Software Technology*, 46(13):899–906.

Urbancová, H. and Linhartová, L. (2011). Staff turnover as a possible threat to knowledge loss. *Journal of competitiveness*, 3(3).

Van der Meulen, R. and Rivera, J. (2014). Gartner says worldwide traditional pc, tablet, ultramobile and mobile phone shipments on pace to grow 7.6 percent in 2014. *Retrieved September*, 29:2015.

Vasilescu, B., Serebrenik, A., Devanbu, P., and Filkov, V. (2014). How social q&a sites are changing knowledge sharing in open source software communities. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 342–354. ACM.

Venzin, M., Von Krogh, G., and Roos, J. (1998). Future research into knowledge management. *Knowing in firms: Understanding, managing and measuring knowledge*, pages 26–66.

Viana, D., Conte, T., Marczak, S., Ferreira, R., and de Souza, C. (2015). Knowledge creation and loss within a software organization: An exploratory case study. In *2015 48th Hawaii International Conference on System Sciences*, pages 3980–3989. IEEE.

Von Krogh, G., Spaeth, S., and Haefliger, S. (2005). Knowledge reuse in open source software: An exploratory study of 15 open source projects. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 198b–198b. IEEE.

Von Krogh, G., Spaeth, S., and Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research policy*, 32(7):1217–1241.

Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S., and Min Tjoa, A. (2007). Monitoring the "health" status of open source web-engineering projects. *International Journal of Web Information Systems*, 3(1/2):116–139.

Wang, X. and Lantzy, S. (2011). A systematic examination of member turnover and online community health.

Wasko, M. M., Faraj, S., et al. (2005). Why should i share? examining social capital and knowledge contribution in electronic networks of practice. *MIS quarterly*, 29(1):35–57.

Williams, L. and Kessler, R. (2002). *Pair programming illuminated*. Addison-Wesley Longman Publishing Co., Inc.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, page 38. Citeseer.

Wohlin, C., Höst, M., and Henningsson, K. (2003). Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer.

Xu, B. (2006). *Volunteers' participative behaviors in open source software development: the role of extrinsic incentive, intrinsic motivation and relational social capital*. PhD thesis, Texas Tech University.

Yager, R. R. and Alajlan, N. (2014). A note on mean absolute deviation. *Inf. Sci.*, 279:632–641.

Ye, Y., Nakakoji, K., Yamamoto, Y., and Kishida, K. (2008). The co-evolution of systems and communities in free and open source software development. In *Global Information Technologies: Concepts, Methodologies, Tools, and Applications*, pages 3765–3776. Igi Global.

Yilmaz, M., Yilmaz, M., O'Connor, R. V., and Clarke, P. (2016). A gamification approach to improve the software development process by exploring the

personality of software practitioners. In *International Conference on Software Process Improvement and Capability Determination*, pages 71–83. Springer.

Yin, R. (2013). *Case study research: Design and methods*. SAGE Publications.

Yu, Y., Benlian, A., and Hess, T. (2012). An empirical study of volunteer members' perceived turnover in open source software projects. In *2012 45th Hawaii International Conference on System Sciences*, pages 3396–3405. IEEE.

Zack, M. H. (1999a). Developing a knowledge strategy. *California management review*, 41(3):125–145.

Zack, M. H. (1999b). Managing codified knowledge. *Sloan management review*, 40(4):45–58.

Zagalsky, A., Teshima, C. G., German, D. M., Storey, M.-A., and Poo-Caamaño, G. (2016). How the r community creates and curates knowledge: a comparative study of stack overflow and mailing lists. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 441–451. ACM.

Zheng, X., Zeng, D., Li, H., and Wang, F. (2008). Analyzing open-source software systems as complex networks. *Physica A: Statistical Mechanics and its Applications*, 387(24):6190–6200.

Zhou, M. and Mockus, A. (2010). Developer fluency: Achieving true mastery in software projects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 137–146. ACM.

Zhou, Q. (2009). The impact of job satisfaction affect on turnover intention: An empirical study based on the circumstances of china. In *2009 Second International Conference on Education Technology and Training*, pages 220–223. IEEE.

Zins, C. (2007). Conceptual approaches for defining data, information, and knowledge. *Journal of the American society for information science and technology*, 58(4):479–493.

Appendix A

Literature Review

A.1 List of Primary Studies

- PS1. Wang, X., Lantzy, S.: A systematic examination of member turnover and online community health. In: ICIS 2011 Proceedings. 25. (2011)
- PS2. Mens, T.: An ecosystemic and socio-technical view on software maintenance and evolution. In: Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on, pp. 1-8. IEEE, (2016)
- PS3. Constantinou, E., Mens, T.: An empirical comparison of developer retention in the RubyGems and npm software ecosystems. Innovations in Systems and Software Engineering 13, 101-115 (2017)
- PS4. Fronza, I., Janes, A., Sillitti, A., Succi, G., Trebeschi, S.: Cooperation wordle using pre-attentive processing techniques. In: 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 57-64. (2013)
- PS5. Joblin, M., Apel, S., Mauerer, W.: Evolutionary trends of developer coordination: A network approach. Empir Software Eng 1-45 (2017)
- PS6. Rashid, M., Clarke, P.M. and O'Connor, R.V., 2017, October. Exploring Knowledge Loss in Open Source Software (OSS) Projects. In International Conference on Software Process Improvement and Capability Determination (pp. 481-495). Springer, Cham.

- PS7. Lin, B., Robles, G., Serebrenik, A.: Developer turnover in global, industrial open source projects: insights from applying survival analysis. In: Proceedings of the 12th International Conference on Global Software Engineering, pp. 66-75. IEEE Press, (2017)

- PS8. Foucault, M., Palyart, M., Blanc, X., Murphy, G.C., Falleri, J.-R.: Impact of developer turnover on quality in open-source software. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 829-841. ACM, Bergamo, Italy (2015)

- PS9. Nidhra, S., Yanamadala, M., Afzal, W., Torkar, R.: Knowledge transfer challenges and mitigation strategies in global software development—A systematic literature review and industrial validation. International Journal of Information Management 33, 333-355 (2013)

- PS10. Huang, P., Zhang, Z.: Participation in Open Knowledge Communities and Job-Hopping: Evidence from Enterprise Software (2016)

- PS11. Rigby, P.C., Zhu, Y.C., Donadelli, S.M., Mockus, A.: Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a project at Avaya. In: Proceedings of the 2016 International Conference on Software Engineering. (2016)

- PS12. Britto, R., Smite, D., Damm, L.-O.: Software Architects in Large-Scale Distributed Projects: An Ericsson Case Study. IEEE Software 33, 48-55 (2016)

- PS13. Izquierdo-Cortazar, D., Robles, G., Ortega, F., Gonzalez-Barahona, J.M.: Using software archaeology to measure knowledge loss in software projects due to developer turnover. In: System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on, pp. 1-10. IEEE, (2009)

- PS14. Ayushi, R., Ashish, S.: What Community Contribution Pattern Says

about Stability of Software Project? In: Software Engineering Conference (APSEC), 2014 21st Asia-Pacific, pp. 31-34. (2014)

- PS15. Bao, L., Xing, Z., Xia, X., Lo, D., Li, S.: Who Will Leave the Company?: A Large-Scale Industry Study of Developer Turnover by Mining Monthly Work Report. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 170-181. (2017)

- PS16. Schilling, A., Laumer, S., Weitzel, T.: Who Will Remain? An Evaluation of Actual Person-Job and Person-Team Fit to Predict Developer Retention in FLOSS Projects. In: System Science (HICSS), 2012 45th Hawaii International Conference on, pp. 3446-3455. (2012)

- PS17. Ransbotham, S., Kane, G.C.: Membership turnover and collaboration success in online communities: Explaining rises and falls from grace in Wikipedia. Mis Quarterly 613-627 (2011)

- PS18. Garcia, D., Zanetti, M.S., Schweitzer, F.: The Role of Emotions in Contributors Activity: A Case Study on the GENTOO Community. In: Cloud and Green Computing (CGC), 2013 Third International Conference on, pp. 410-417. (2013)

- PS19. Bosu, A., Carver, J.C.: Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation. Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1-10. ACM, Torino, Italy (2014)

- PS20. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology 11, 309-346 (2002)

- PS21. Robles, G., Gonzalez-Barahona, J.M.: Contributor turnover in libre software projects. IFIP International Federation for Information Processing, vol. 203, pp. 273-286 (2006)

- PS22. Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S., Min Tjoa, A.: Monitoring the "health" status of open source web-engineering projects. International Journal of Web Information Systems 3, 116-139 (2007)

- PS23. Otte, T., Moreton, R., Knoell, H.D.: Applied quality assurance methods under the open source development model. In: Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International, pp. 1247-1252. (2008)

- PS24. Crowston, K., Wei, K., Howison, J., Wiggins, A.: Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. Acm Computing Surveys 44, (2012)

- PS25. Capiluppi, A., Gonzalez-Barahona, J.M., Herraiz, I., Robles, G.: Adapting the "staged model for software evolution" to free/libre/open source software. Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting, pp. 79-82. ACM, Dubrovnik, Croatia (2007)

- PS26. Vasilescu, B., Serebrenik, A., Devanbu, P., Filkov, V.: How social Q&A sites are changing knowledge sharing in open source software communities. Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing, pp. 342-354. ACM, Baltimore, Maryland, USA (2014)

- PS27. Nonaka, I., Toyama, R., Konno, N.: SECI, Ba and Leadership: a Unified Model of Dynamic Knowledge Creation. Long Range Planning 33, 5-34 (2000)

- PS28. Robles, G., Gonzalez-Barahona, J.M., Michlmayr, M.: Evolution of volunteer participation in libre software projects: evidence from Debian. In: Proceedings of the 1st International Conference on Open Source Systems, pp. 100-107. (2005)

- PS29. Rigby, P.C., German, D.M., Cowen, L., Storey, M.-A.: Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory. ACM Trans. Softw. Eng. Methodol. 23, 1-33 (2014)

- PS30. Herraiz, I., Robles, G., Jose, J., A., Romera, T., Jesus, Gonza lez.M., Barahona,.: The processes of joining in global distributed software projects. Proceedings of the 2006 international workshop on Global software development for the practitioner, pp. 27-33. ACM, Shanghai, China (2006)

- PS31. Steinmacher, I., Silva, M.A.G., Gerosa, M.A., Redmiles, D.F.: A systematic literature review on the barriers faced by newcomers to open source software projects. Information and Software Technology 59, 67-85 (2015)

- PS32. Licorish, S.A., MacDonell, S.G.: Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. Information and Software Technology 56, 1578-1596 (2014)

- PS33. Lee, G.K., Cole, R.E.: From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development. Organization Science 14, 633-649 (2003)

- PS34. Qin, X., Salter-Townshend, M., Cunningham, P.: Exploring the Relationship between Membership Turnover and Productivity in Online Communities. In: ICWSM. (2014)

- PS35. Zagalsky, A., Teshima, C.G., German, D.M., Storey, M.-A., Poo-Caamaño, G.: How the R community creates and curates knowledge: a comparative study of stack overflow and mailing lists. In: Proceedings of the 13th International Workshop on Mining Software Repositories, pp. 441-451. ACM, (2016)

- PS36. Sowe, S.K., Stamelos, I., Angelis, L.: Understanding knowledge sharing activities in free/open source software projects: An empirical study. Journal of Systems and Software 81, 431-446 (2008)

- PS37. Kuk, G.: Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List. Manage. Sci. 52, 1031-1042 (2006)

- PS38. Chen, X., Li, X., Clark, J.G., Dietrich, G.B.: Knowledge sharing in open source software project teams: A transactive memory system perspective. International Journal of Information Management 33, 553-563 (2013)

Appendix B

Proactive Knowledge Retention Canonical Model Development

B.1 Master Table Containing Data Components Linked to data Sources

Data Component No.	Data Components	OSS Literature	Organisations	Others	Online Open Source Guides +
1	Visualisation of Resources- A visualization word cloud has been proposed to show quickly the level of cooperation of the team in the project [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors.	**x**			
2	A successor is a person who has relevant expertise and is knowledgeable on the work of other contributors. Identification of successors and involving them as co-owners is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10]	**x**			
3	Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95].	**x**			

4	Centralisation: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams.	**x**		
5	Removal of Knowledge Barriers: Only a few OSS contributors transit to a higher learning state, due to high learning barriers. Consequently, it can take newcomers up to 60 weeks to become an effective contributor to a OSS project [129]. Knowledge retention in OSS projects can also be improved by the removal of knowledge barriers: namely, lack of technical experience, lack of domain expertise and lack of project practices that hinder the contributions [130, 131].	**x**		
6	An insight into the area of expertise members, a knowledge map or directory can be used on the project website internal knowledge portal, serving as a way to help knowledge workers familiarize themselves with their colleagues knowledge, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge	**x**		
7	Transactive Memory System- development within the teams, based on knowledge location, the usage of the developer mailing list and knowledge credibility	**x**		

8	Diversity of Core Contributors: For an OSS project to survive, a diversity of core developers is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant imbalance between the contribution and the response from the developers' community was noticed. The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next.	**x**		
9	Uniform Knowledge Distribution: The communication of the OSS project is directed by the core contributors. Their attitudes and involvement in knowledge sharing were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities. The more code changes core developers perform, the more knowledge they provide [124]. However, their least involvement in communication and task changes results into some negative team attitudes.	**x**		
10	This kind of disruption in communication in OSS projects can hinder knowledge sharing. Another resolution to the non-uniform distribution of knowledge may be the proactive assignment of maintenance tasks on the code written by other contributors. As indicated that contributors who modify codes from other contributors stay longer on the project [28]. This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.	**x**		

11	Gamification: A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133].	**x**		
12	Improving Code Review Feedback Time for non-Cores: Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors [134]. As indicated by a social network analysis of the code review data from eight popular OSS projects, core developers as compared to peripheral contributors have the benefit of receiving quicker feedback, face shorter review intervals and have a higher code acceptance rate [135]. Due to the lack of an established reputation, peripheral developers wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process. Accordingly, a delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor [135]. An improvement to the timings of the review feedbacks in OSS projects to peripherals can result in a uniform distribution of knowledge, reduce the effects of turnover, and motivate newcomers to stay for a longer duration.	**x**		

13	Code review checklists also provide team members with clear expectations for each type of review			**x**
14	Use of a collaborative code review tool that allows reviewers to log bugs, discuss them with the author, and approve changes in the code			**x**
15	Guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options	**x**		
16	Integrators should be proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette. Integrator and contributors should agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub)			**x**
17	Projects should provide a policy or comprehensive set of contribution guidelines			**x**
18	Good documentation invites people to interact with project open an issue or pull request. Use these interactions as opportunities to move them down the funnel (from users to contributors and to maintainers)			**x**
19	open-minded about the types of contributions to accept start with a bug report or small fix make easy for casual contributors to contribute document your process with an access to public			**x**
20	Transparency about project's roadmap, the types of contributions you're looking for, how contributions are reviewed, or why you made certain decisions			**x**

21	communication is public and accessible, anybody can read past archives to get up to speed and participate. Merged with No. 18			**x**
22	When multiple users running into the same problem, document the answers in the README			**x**
23	Write down your project's vision, a project roadmap,make those public as well, Add them to your README, or create a separate file called VISION.			**x**
24	While working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on.			**x**
25	to be responsive when someone files an issue, submits a pull request, or asks a question about your project. Responding quickly, people will feel they are part of a dialogue, and they'll be more enthusiastic about participating set up notifications in some of these places (such as Stack Overflow, Twitter, or Reddit) so you are alerted when someone mentions your project			**x**
26	negative people will make other people in your community uncomfortable, a supportive community is the key			**x**
27	label bugs that are suitable for different types of contributors: for example, "first timers only", "good first issue", or "documentation". These labels make it easy for someone new to your project to quickly scan your issues and get started. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors			**x**

28	Share ownership of your project, if you need to step away from your project, either on hiatus or permanently, there's no shame in asking someone else to take over for you. Find support for your users and community while you're away from a project. If you can't find the support you need, take a break anyway. Be sure to communicate when you're not available, so people aren't confused by your lack of responsiveness.			**x**
29	CONTRIBUTORS or AUTHORS file in your project that lists everyone who's contributed to your project.			**x**
30	Appreciation...newsletter or write a blog post thanking contributors //			**x**
31	Non- restrictive commit access-Give every contributor commit access, this made people more excited to polish their patches			**x**
32	Resolving conflicts			**x**
33	Emphasize "consensus seeking" rather than consensus. community members discuss major concerns until they feel they have been adequately heard			**x**
34	keep your documentation up-to-date			**x**
35	Be proactive to reduce the volume of unwanted contributions in the first place, explain your project's process for submitting and accepting contributions in your contributing guide. Fill out a issue or PR template/checklist and open an issue before submitting a PR			**x**
36	Someone is enthusiastic about your project, but needs a bit of polish consider mentoring them through their first contribution		**x**	**x**

37	Encouraging your community members to work on their own fork can provide the creative outlet they need, without conflicting with your project's vision.			**x**
38	Require tests and other checks to improve the quality of your code. If you add tests, make sure to explain how they work in your CONTRIBUTING file.			**x**
39	Designating leaders can be as simple as adding their names to your README or a CONTRIBUTORS text file.			**x**
40	Let people self-organize and volunteer for the roles they're most excited about, rather than assigning them.			**x**
41	Once you've established leadership roles, don't forget to document how people can attain them! Establish a clear process for how someone can become a maintainer or join a subcommittee in your project, and write it into your GOVERNANCE.md.			**x**
42	Under a liberal contribution model, the people who do the most work are recognized as most influential, but this is based on current contributions. Major project decisions are made based on a consensus seeking process work and not historic			**x**
43	A "core team" of maintainers with specific areas, subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct).			**x**
44	Governance file to established leadership roles and a clear process for how someone can become a maintainer or join a subcommittee in project, and document it into your GOVERNANCE.md.			**x**

45	Post mortem review or after action review to learn through a collective activity and to build knowledge based on the experience to improve future practice. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects		**x**	
46	A Community of Practice to create a network to collect and exchange information about the knowledge-transfer methods to diffuse knowledge before it is lost		**x**	
47	Written reports to transfer project relevant knowledge with other contributors		**x**	
48	Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices, such as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation.		**x**	
49	Using story telling to generating, share, and discuss stories for quickly integrating new learning		**x**	

50	Experience Based Memory: Experience packaged and stored in an experience base built by contributors based on their experiences including resources such as all experience types, lesson learned, project data, and technology reports. Version control, change management, documenting design decisions, and requirements traceability are software engineering practices that help build project and product memories as an indirect or direct effect of software development.		**x**	
51	Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing		**x**	
52	Interviews in an organisation is a knowledge transfer knowledge process used to integrate the knowledge captured into the organisation		**x**	
53	Recognition and Reward Structure: In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the organisation. Furthermore, to encourage contribution of knowledge, based on codification and personalisation a reward system is established for people documenting and sharing knowledge (Hansen et al. 1999). A reward structure is based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition. Extrinsic motivation is related to monetary incentives (Gammelgaard 2007). Organisations like Xerox, and Hewlett-Packard reward people for sharing their knowledge (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003).		**x**	

	Using the combination of extrinsic and intrinsic rewards is better (Gammelgaard 2007). In the two types of reward structures, the long lasting one is intrinsic reward structure. As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated. Software engineers willingly share their knowledge. Even though the technology changes very quickly, capturing the gained knowledge still is worth the effort (Rus and Lindvall 2002).			
54	Relevant technical tools allow for the retention of supporting documents, and competence management and used by software organisations for supporting software engineering practices along with knowledge management. For example, document management tools frequently employed in organisations are Hyperwave, Microsoft SharePoint, Lotus Domino, and Xerox DocuShare.		**x**	

B.2 Merging Conceptual Duplicates

Data Component	Data Components * Candidate rows for merging highlighted in Green	Memo	Merged Duplicates Highlighted in Colour
1	Visualisation of Resources- A visualization word cloud has been proposed to show quickly the level of cooperation of the team in the project [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors.		**Visualisation of Resources tool**- A visualization word cloud has been proposed **to show quickly the level of cooperation of the team in the project** [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors. // **Technology Oriented Tools**
2	A successor is a person who has relevant expertise and is knowledgeable on the work of other contributors. Identification of successors and involving them as co-owners is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10]		**Identifying successors with relevant expertise and are knowledgeable on the work of other contributors.** Identification of successors and involving contributors as co-owners with relevant expertise knowledgeable on the work of other contributors. is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10] // **Knowledge transfer and Sharing**

3	Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95].		Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. such as pair programming, facilitate knowledge sharing between peers // **Shared code knowledge - pair programming**
4	Centralisation: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams.		**Centralisation**: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams.
5	Removal of Knowledge Barriers: Only a few OSS contributors transit to a higher learning state, due to high learning barriers. Consequently, it can take newcomers up to 60 weeks to become an effective contributor to a OSS project [129]. Knowledge retention in OSS projects can also be improved by the removal of knowledge barriers: namely, lack of technical experience, lack of domain expertise and lack of project practices that hinder the contributions [130, 131].		**Removal of Knowledge Barriers:** Only a few OSS contributors transit to a higher learning state, due to high learning barriers. Consequently, it can take newcomers up to 60 weeks to become an effective contributor to a OSS project [129]. Knowledge retention in OSS projects can also be improved by the removal of knowledge barriers: namely, lack of **technical experience**, lack of **domain expertise** and lack of **knowledge on project practices** that hinder the contributions [130, 131]. // **Contributions**

6	An insight into the area of expertise members, a knowledge map or directory can be used on the project website internal knowledge portal, serving as a way to help knowledge workers familiarize themselves with their colleagues knowledge, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks		An insight into the area of expertise members, **a knowledge map or directory** can be used **on the project website** internal knowledge portal, serving as a way **to help knowledge workers familiarize** themselves **with their colleagues knowledge**, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks. // **Technology Oriented toolsAn insight into the area of expertise members, a knowledge map or directory** can be used **on the project website** internal knowledge portal, serving as a way **to help knowledge workers familiarize** themselves **with their colleagues knowledge**, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks.
7	Transactive Memory System-development within the teams, based on knowledge location, the usage of the developer mailing list and knowledge credibility		**Transactive Memory System-**development within the teams, **based on knowledge location, the usage of the developer mailing list and knowledge credibility // Technology Oriented tools**

8	Diversity of Core Contributors: For an OSS project to survive, a diversity of core developers is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant imbalance between the contribution and the response from the developers' community was noticed. The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next.		**Diverse specialisations of Core Contributors**: For an OSS project to survive, a **diversity of core developers** is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant **imbalance between the contribution and the response from the developers' community was noticed.** The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next. // **Balance between the contribution submitted and response from specialised core developers in community. A "core team" of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct).**

9	Uniform Knowledge Distribution: The communication of the OSS project is directed by the core contributors. Their attitudes and involvement in knowledge sharing were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities. The more code changes core developers perform, the more knowledge they provide [124]. However, their least involvement in communication and task changes results into some negative team attitudes.		**Uniform Knowledge Distribution:** The **communication of the OSS project is directed by the core contributors**. Their **attitudes and involvement in knowledge sharing** were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. **They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities.** The more code changes core developers perform, the more knowledge they provide [124]. However, their least involvement in communication and task changes results into some negative team attitudes. // **Knowledge communication from cores to non-cores**
10	This kind of disruption in communication in OSS projects can hinder knowledge sharing. Another resolution to the non-uniform distribution of knowledge may be the proactive assignment of maintenance tasks on the code written by other contributors. As indicated that contributors who modify codes from other contributors stay longer on the project [28]. This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.		This kind of disruption in communication in OSS projects can hinder knowledge sharing. Another resolution to the non-uniform distribution of knowledge may be the **proactive assignment of maintenance tasks on the code written by other contributors.** As indicated that contributors who modify codes from other contributors stay longer on the project [28]. This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.

11	Gamification: A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133].		Gamification: A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133].

12	Improving Code Review Feedback Time for non-Cores: Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors [134]. As indicated by a social network analysis of the code review data from eight popular OSS projects, core developers as compared to peripheral contributors have the benefit of receiving quicker feedback, face shorter review intervals and have a higher code acceptance rate [135]. Due to the lack of an established reputation, peripheral developers wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process. Accordingly, a delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor [135]. An improvement to the timings of the review feedbacks in OSS projects to peripherals can result in a uniform distribution of knowledge, reduce the effects of turnover, and motivate newcomers to stay for a longer duration.		**Improving Code Review Feedback Time for non-Cores:** Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors [134]. As indicated by a social network analysis of the code review data from eight popular OSS projects, **core developers as compared to peripheral contributors have the benefit of receiving quicker feedback, face shorter review intervals and have a higher code acceptance rate** [135]. Due to the **lack of an established reputation, peripheral developers wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process**. Accordingly, a **delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor** [135]. An improvement to the timings of the review feedbacks in OSS projects to peripherals can result in a **uniform distribution of knowledge**, reduce the effects of turnover, and motivate newcomers to stay for a longer duration.
13	Code review checklists also provide team members with clear expectations for each type of review		**Code review checklists also provide team members with clear expectations for each type of review**
14	Use of a collaborative code review tool that allows reviewers to log bugs, discuss them with the author, and approve changes in the code		**Use of a collaborative code review tool** that allows reviewers **to log bugs, discuss them with the author, and approve changes in the code**.

15	Guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options	Merged with No. 18	**Integrators** should be **proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette.** Integrator and contributors should **agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels** (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub)
16	Integrators should be proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette. Integrator and contributors should agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub)		**Good documentation** invites people to interact with project open an issue or pull request. Use these interactions as opportunities to move them down the funnel (from users to contributors and to maintainers). **communication is public and accessible, anybody can read past archives to get up to speed and participate. Guidelines should at least provide details about the expeced code style, commit format, pull request process, and available communication options. Projects should provide a policy or comprehensive set of contribution guidelines. Write down your project's vision, a project roadmap,make those public as well, Add them to your README, or create a separate file called VISION. keep your** **documentation up-to-date. Be**

			proactive to reduce the volume of unwanted contributions in the first place, explain your project's process for submitting and accepting contributions in your contributing guide. Fill out a issue or PR template/checklist and open an issue before submitting a PR.
17	Projects should provide a policy or comprehensive set of contribution guidelines	Merged with No. 18	**open-minded** about the types of contributions to accept start with a bug report or small fix make easy for casual contributors to contribute. document your process with an access to public
18	Good documentation invites people to interact with project open an issue or pull request. Use these interactions as opportunities to move them down the funnel (from users to contributors and to maintainers)		**Transparency about project's roadmap**, the **types of contributions you're looking for**, how contributions are reviewed, or why you made certain decisions
19	open-minded about the types of contributions to accept start with a bug report or small fix make easy for casual contributors to contribute document your process with an access to public		When multiple users running into the same problem, document the answers in the README
20	Transparency about project's roadmap, the types of contributions you're looking for, how contributions are reviewed, or why you made certain decisions		While working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on. **//Encouraging contribution**

21	communication is public and accessible, anybody can read past archives to get up to speed and participate. Merged with No. 18	Merged with No. 18	to be **responsive when someone files an issue, submits a pull request, or asks a question about your project**. Responding quickly, people will feel they are part of a dialogue, and they'll be more enthusiastic about participating set up notifications in some of these places (such as Stack Overflow, Twitter, or Reddit) so you are alerted when someone mentions your project. if you can't review the request immediately, **acknowledging it early helps increase engagement.**
22	When multiple users running into the same problem, document the answers in the README		negative people will make other people in your community uncomfortable, a supportive community is the key, **resolving conflicts** //**Community Health**
23	Write down your project's vision, a project roadmap,make those public as well, Add them to your README, or create a separate file called VISION.	Merged with No. 18	label bugs that are suitable for different types of contributors: for example, "first timers only", "good first issue", or "documentation". These labels make it easy for someone new to your project to quickly scan your issues and get started. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors
24	While working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on.		**Share ownership of** your **project**, if you need to step away from your project, either on hiatus or permanently, there's no shame in asking someone else to take over for you. Find **support for your users and community while you're away from a project**. If you can't find the support you need, take a break anyway. Be sure to communicate when you're not available, so people aren't confused by your lack of responsiveness.

25	to be responsive when someone files an issue, submits a pull request, or asks a question about your project. Responding quickly, people will feel they are part of a dialogue, and they'll be more enthusiastic about participating set up notifications in some of these places (such as Stack Overflow, Twitter, or Reddit) so you are alerted when someone mentions your project		newsletter or write a blog post thanking contributors //**Appreciation**
26	negative people will make other people in your community uncomfortable, a supportive community is the key		Non- restrictive commit access - Give every contributor commit access, this made people more excited to polish their patches
27	label bugs that are suitable for different types of contributors: for example, "first timers only", "good first issue", or "documentation". These labels make it easy for someone new to your project to quickly scan your issues and get started. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors		Emphasize "consensus seeking" rather than consensus. community members discuss major concerns until they feel they have been adequately heard// **Governance**
28	Share ownership of your project, if you need to step away from your project, either on hiatus or permanently, there's no shame in asking someone else to take over for you. Find support for your users and community while you're away from a project. If you can't find the support you need, take a break anyway. Be sure to communicate when you're not available, so people aren't confused by your lack of responsiveness.		Someone is enthusiastic about your project, but needs a bit of polish consider mentoring them through their first contribution. mentor someone who'd like to contribute

29	CONTRIBUTORS or AUTHORS file in your project that lists everyone who's contributed to your project.	Merged with No. 40	Encouraging your community members to work on their own fork can provide the creative outlet they need, without conflicting with your project's vision
30	Appreciation...newsletter or write a blog post thanking contributors //		Require tests and other checks to improve the quality of your code. If you add tests, make sure to explain how they work in your CONTRIBUTING file
31	Non- restrictive commit access-Give every contributor commit access, this made people more excited to polish their patches		Designating leaders can be as simple as adding their names to your README or a CONTRIBUTORS text file. **CONTRIBUTORS or AUTHORS file in your project that lists everyone who's contributed to your project** // **Documentation**
32	Resolving conflicts	Merged with 26	Let people self-organize and volunteer for the roles they're most excited about, rather than assigning them.
33	Emphasize "consensus seeking" rather than consensus. community members discuss major concerns until they feel they have been adequately heard		Once you've established leadership roles, don't forget to document how people can attain them! **Establish a clear process for how someone can become a maintainer or join a subcommittee in your project, and write it into your GOVERNANCE.md**
34	keep your documentation up-to-date	Merged with No. 18	Under a liberal contribution model, the people who do the most workare recognized as most influential, but this is based on current contributions. Major project decisions are made based on a consensus seeking processwork and not historic

35	Be proactive to reduce the volume of unwanted contributions in the first place, explain your project's process for submitting and accepting contributions in your contributing guide. Fill out a issue or PR template/checklist and open an issue before submitting a PR	Merged with No. 18	**Post mortem review or after action review** to learn through a collective activity and **to build knowledge based on the experience to improve future practice**. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects.
36	Someone is enthusiastic about your project, but needs a bit of polish consider mentoring them through their first contribution		A Community of Practice to create a network to collect and exchange information about the knowledge-transfer methods to diffuse knowledge before it is lost
37	Encouraging your community members to work on their own fork can provide the creative outlet they need, without conflicting with your project's vision		Written reports to transfer project relevant knowledge with other contributors
38	Require tests and other checks to improve the quality of your code. If you add tests, make sure to explain how they work in your CONTRIBUTING file		Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices,while **job rotation helps knowledge spread throughout the project or organisation.**
39	Designating leaders can be as simple as adding their names to your README or a CONTRIBUTORS text file.		Using story telling to generating, share, and discuss stories for quickly integrating new learning.

40	Let people self-organize and volunteer for the roles they're most excited about, rather than assigning them.		**Experience Based Memory**: Experience packaged and stored in an experience base **built by contributors** based **on** their experiences including **resources such as all experience types, lesson learned, project data, and technology reports**. **Version control, change management, documenting design decisions, and requirements traceability** are software engineering practices that help build project and product memories as an indirect or direct effect of software development. **Relevant technical tools allow for the retention of supporting documents, and competence management and used by software organisations for supporting software engineering practices along with knowledge management. For example, document management tools frequently employed in organisations are Hyperwave, Microsoft SharePoint, Lotus Domino, and Xerox DocuShare. Technology based systems// Technology oriented tools**
41	Once you've established leadership roles, don't forget to document how people can attain them! Establish a clear process for how someone can become a maintainer or join a subcommittee in your project, and write it into your GOVERNANCE.md		Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing.
42	Under a liberal contribution model, the people who do the most work are recognized as most influential, but this is based on current contributions. Major project decisions are made based on a consensus seeking process work and not historic		Interviews in an organisation is a knowledge transfer knowledge process used to integrate the knowledge captured into the organisation

43	A "core team" of maintainers with specific areas, subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct).	Merged with No 8.	**Rewarding contributors for knowledge sharing and transfer with recognition and appreciation for best answer provided.** Recognition and Reward Structure: In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the organisation. Furthermore, to encourage contribution of knowledge, based on codification and personalisation **a reward system is established for people documenting and sharing knowledge** (Hansen et al. 1999). A reward structure is based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition. Extrinsic motivation is related to monetary incentives (Gammelgaard 2007). Organisations like Xerox, and Hewlett-Packard **reward people for sharing their knowledge** (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003). Using the combination of extrinsic and intrinsic rewards is better (Gammelgaard 2007). In the two types of reward structures, the long lasting one is intrinsic reward structure. As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being

			compensated. Software engineers willingly share their knowledge. Even though the technology changes very quickly, capturing the gained knowledge still is worth the effort (Rus and Lindvall 2002).
44	Governance file to established leadership roles and a clear process for how someone can become a maintainer or join a subcommittee in project, and document it into your GOVERNANCE.md.	Merged with No. 42	
45	Post mortem review or after action review to learn through a collective activity and to build knowledge based on the experience to improve future practice. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects.		
46	A Community of Practice to create a network to collect and exchange information about the knowledge-transfer methods to diffuse knowledge before it is lost		
47	Written reports to transfer project relevant knowledge with other contributors		

48	Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices, such as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation.	
49	Using story telling to generating, share, and discuss stories for quickly integrating new learning	
50	Experience Based Memory: Experience packaged and stored in an experience base built by contributors based on their experiences including resources such as all experience types, lesson learned, project data, and technology reports. Version control, change management, documenting design decisions, and requirements traceability are software engineering practices that help build project and product memories as an indirect or direct effect of software development.	
51	Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing	
52	Interviews in an organisation is a knowledge transfer knowledge process used to integrate the knowledge captured into the organisation	

53	Recognition and Reward Structure: In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the organisation. Furthermore, to encourage contribution of knowledge, based on codification and personalisation a reward system is established for people documenting and sharing knowledge (Hansen et al. 1999). A reward structure is based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition. Extrinsic motivation is related to monetary incentives (Gammelgaard 2007). Organisations like Xerox, and Hewlett-Packard reward people for sharing their knowledge (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003). Using the combination of extrinsic and intrinsic rewards is better (Gammelgaard 2007). In the two types of reward structures, the long lasting one is intrinsic reward structure. As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated. Software engineers willingly share their knowledge. Even though the technology changes very quickly,	

	capturing the gained knowledge still is worth the effort (Rus and Lindvall 2002).	
54	Relevant technical tools allow for the retention of supporting documents, and competence management and used by software organisations for supporting software engineering practices along with knowledge management. For example, document management tools frequently employed in organisations are Hyperwave, Microsoft SharePoint, Lotus Domino, and Xerox DocuShare.	Merged with No. 51

B.3 Primary Classification of Data Components

Data Component No.	Data Components * Two added practices by researcher highlighted in yellow	Memo No:	Primary Classification and Memo Description
1	**Visualisation of Resources tool**- A visualization word cloud has been proposed **to show quickly the level of cooperation of the team in the project** [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors. // **Technology Oriented Tools**	1	Solution that invove technology to support knowledge relavant activites in OSS projects. All such practices that can be used as a rsolution to deal with knowledge loss due to contributor turnover are gathered under one **category "Technology oriented tools". Practice -> Utilising Technology Oriented tools**
2	**Identifying successors with relevant expertise and are knowledgeable on the work of other contributors.** Identification of successors and involving contributors as co-owners with relevant expertise knowledgeable on the work of other contributors. is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10] // **Knowledge transfer and Sharing**	2	**Category-> Knowledge transfer and Sharing->** **Practice->Identification of successors as co-owners with relevant knowledge on the work of others.**
3	Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. such as pair programming, facilitate knowledge sharing between peers // **Shared code knowledge - pair programming**	3	**Category-> Knowledge transfer and Sharing->** **Practice-> Shared code knowledge - pair programming**
4	**Centralisation**: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams.	4	Stability of architectural knowledge here means the maintence or updated knowledge in Ericsson during the recurrent movement of people in and out of the product with changing business nees. In survey question rather than using word stable, word maintain is used to avoid confusion of the participants. Because knowledge is always evloving and is never stable. **Centralisation approach to maintain architectural knowledge and its availability to new developers and teams Stability of architectural knowledge here means the maintence or updated knowledge in Ericsson during the recurrent movement of people in and out of the product with changing business nees. In survey question rather than using word stable, word maintain is used to avoid confusion of the participants. Because knowledge is always evloving and is never stable. Practice -> Centralisation**

5	**Removal of Knowledge Barriers:** Only a few OSS contributors transit to a higher learning state, due to high learning barriers. Consequently, it can take newcomers up to 60 weeks to become an effective contributor to a OSS project [129]. Knowledge retention in OSS projects can also be improved by the removal of knowledge barriers: namely, lack of **technical experience**, lack of **domain expertise** and lack of **knowledge on project practices** that hinder the contributions [130, 131]. // **Contributions**	5	Removal of knowledge barriers is useful to new contributors on the project to acquire knowledge and to improve the productivity on the project. The knowledge barriers are lower for people who have more experience on the project. So this implies mainly to new commers on the project. category-> Contributions-> Practice-> removal of knowledge barriers-> Improvement in lack of technical experience, lack of domain expertise and lack of knowledge on project practices that hinder the contributions.
6	An insight into the area of expertise members, **a knowledge map or directory** can be used **on the project website** internal knowledge portal, serving as a way **to help knowledge workers familiarize** themselves **with their colleagues knowledge**, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks. // **Technology Oriented toolsAn insight into the area of expertise members, a knowledge map or directory** can be used **on the project website** internal knowledge portal, serving as a way **to help knowledge workers familiarize** themselves **with their colleagues knowledge**, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks.	6	memo # 1
7	**Transactive Memory System**- development within the teams, **based on knowledge location, the usage of the developer mailing list and knowledge credibility // Technology Oriented tools**	7	memo # 1
8	**Diverse specialisations of Core Contributors**: For an OSS project to survive, a **diversity of core developers** is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant **imbalance between the contribution and the response from the developers' community was noticed.** The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next. // **Balance between the contribution submitted and response from specialised core developers in community. a "core team" of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct).**	8	Diversity of core contributors relate to a set of contributors with diverse specialisation. Lack of these contributors delays required feedback on the submitted contribution by non-cores. The main category is -> Uniform knowledge distribution-> Practice-> diverse specialisation of core contributors-> details subcategory -> attaining balance between the contribution submitted and response from specialised core contributors in community

9	**Uniform Knowledge Distribution:** The **communication of the OSS project is directed by the core contributors**. Their **attitudes and involvement in knowledge sharing** were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. **They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities.** The more code changes core developers perform, the more knowledge	9	refer to memo #8, The main category is -> Uniform knowledge distribution-> Practice-> Knowledge communication from Cores to non- cores -> start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities. Core control the flow of information and knowledge to non cores
10	This kind of disruption in communication in OSS projects can hinder knowledge sharing. Another resolution to the non-uniform distribution of knowledge may be the **proactive assignment of maintenance tasks on the code written by other contributors.** As indicated that contributors who modify codes from other contributors stay longer on the project [28]. This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.	10	refer to memo #8, The main category is -> Uniform knowledge distribution-> Practice-> proactive assignment of tasks to non-cores-> proactive assignment of maintenance tasks on the code written by other contributors to equally develop skills of non-cores
11	Gamification: A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133].	11	refer to memo # 45, Category->Extrinsic motivation-> Practice->Rewards for knowledge sharing and transfer category->Using gamification qualities to evaluate who provides the best answer is the one that gains the most points
12	Assessing contributors on the meritocracy level for knowledge sharing and transfer similar to the one followed for code contributions on the project	12	refer to memo # 45, Extrinsic motivation-> Rewards for knowledge sharing and transfer->Practice-> Assessing meritocracy level of contributors on knowledge sharing and transfer similar to the one followed for code contributions on the project

13	**Improving Code Review Feedback Time for non-Cores:** Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors [134]. As indicated by a social network analysis of the code review data from eight popular OSS projects, **core developers as compared to peripheral contributors have the benefit of receiving quicker feedback, face shorter review intervals and have a higher code acceptance rate** [135]. Due to the **lack of an established reputation, peripheral developers wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process**. Accordingly, a **delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor** [135]. An improvement to the timings of the review feedbacks in OSS projects to peripherals can result in a **uniform distribution of knowledge**, reduce the effects of turnover, and motivate newcomers to stay for a longer duration.	13	refer to memo # 5. In addition to be a source of negative motivation for new contributors or peripherals, delay in code feedback time is also a barrier to knowledge contributions. Category-> contributions-> Practice-> removal of knowledge sharing barriers->Improving code feedback time for non-cores
14	Project policy to **invite contributors from all roles to participate in peer review**s to transfer project relevant knowledge and experience	14	**Category-> Uniform Knowledge Distribution->** **Practice-> invite contributors from all roles to participate in peer reviews**
15	**Code review checklists also provide team members with clear expectations for each type of review**	15	**refer to memo # 5. Category-> contributions-> Practice-> Project rules and policies-> Code review checklists for team members with clear expectations for each type of review contributions**
16	**Use of a collaborative code review tool** that allows reviewers **to log bugs, discuss them with the author, and approve changes in the code**.	16	**refer to memo # 5. Category-> contributions-> Practice-> Project rules and policies-> use of collaborative review tool to log bugs, discuss them with the author, and approve changes in the code.**
17	**Integrators** should be **proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette.** Integrator and contributors should **agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels** (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub)	17	**refer to memo # 5. Category-> contributions-> Practice-> Project rules and policies-> proactively establishing a professional communication etiquette and minimal communication protocols for mandatory information exchange -> communication is public and accessible, past archives can be read to get up to speed and participate**

18	**Good documentation** invites people to interact with project open an issue or pull request. Use these interactions as opportunities to move them down the funnel (from users to contributors and to maintainers). **communication is public and accessible, anybody can read past archives to get up to speed and participate. Guidelines should at least provide details about the expeced code style, commit format, pull request process, and available communication options. Projects should provide a policy or comprehensive set of contribution guidelines. Write down your project's vision, a project roadmap,make those public as well, Add them to your README, or create a separate file called VISION. keep your documentation up-to-date. Be proactive to reduce the volume of unwanted contributions in the first place, explain your project's process for submitting and accepting contributions in your contributing guide. Fill out a issue or PR template/checklist and open an issue before submitting a PR.**	18	**Category-> contributions-> Practice-> Project rules and policies-> Good updated documentation for contributors to interact with project and open pull request and progress in project**
19	**open-minded** about the types of contributions to accept start with a bug report or small fix make easy for casual contributors to contribute. document your process with an access to public	19	**Category-> contributions-> Practice-> Project rules and policies->Open to all kinds of contributions on the project ->Public access is provided during the documentation of project process**
20	**Transparency about project's roadmap**, the **types of contributions you're looking for**, how contributions are reviewed, or why you made certain decisions	20	**Category-> contributions-> Practice-> Project rules and policies-> Transparency about project's roadmap, the types of contributions required, review process, and rational on decision making**
21	When multiple users running into the same problem, document the answers in the README	21	**Category-> Knowledge Sharing->Practice->Standard Documentation->When multiple users run into the same problem, document the answers in the README**
22	While working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on. **//Encouraging contribution**	22	**Category-> contributions-> Practice-> removal of knowledge sharing barriers-> While working on a substantial update status as work in progress (WIP) for other people to feel involved in the process early on.**
23	to be **responsive when someone files an issue, submits a pull request, or asks a question about your project**. Responding quickly, people will feel they are part of a dialogue, and they'll be more enthusiastic about participating set up notifications in some of these places (such as Stack Overflow, Twitter, or Reddit) so you are alerted when someone mentions your project. if you can't review the request immediately, **acknowledging it early helps increase engagement.**	23	**Category-> contributions-> Practice-> removal of knowledge sharing barriers-> Acknowledging or to be responsive when someone files an issue, submits a code, asks questions.**

24	negative people will make other people in your community uncomfortable, a supportive community is the key, resolving conflicts //**Community Health**	24	**Category-> Community Health-> supportive community to resolve conflicts**
25	label bugs that are suitable for different types of contributors: for example, “first timers only”, “good first issue”, or “documentation”. These labels make it easy for someone new to your project to quickly scan your issues and get started. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors	25	**Category-> contributions-> Practice-> removal of knowledge sharing barriers-> label bugs that are suitable for different types of contributors as opportunities to recruit new contributors**
26	**Share ownership of** your **project**, if you need to step away from your project, either on hiatus or permanently, there’s no shame in asking someone else to take over for you. Find **support for your users and community while you’re away from a project**. If you can’t find the support you need, take a break anyway. Be sure to communicate when you’re not available, so people aren’t confused by your lack of responsiveness.	26	Governance and Leadership practices to continue the project evolving through maintenance when the owner or the main contributor steps away Sharing ownership of the project can be considered a Practice of governance and leadership. Governance and Leadership -> Practice->Sharing project ownership to support user community in absence of project owners.
27	newsletter or write a blog post thanking contributors //**Appreciation**	27	**Intrinsic motivation->Practice-> appreciation as newsletter or a blog post thanking contributors**
28	Non- restrictive commit access - Give every contributor commit access, this made people more excited to polish their patches	28	**Governance and Leadership -> Practice->Non- restrictive commit access - Give every contributor commit access to polish their patches**
29	Emphasize “consensus seeking” rather than consensus. community members discuss major concerns until they feel they have been adequately heard// **Governance**	29	**Governance and Leadership -> Practice-> Emphasize “consensus seeking” in community to discuss major concerns until adequately heard**
30	Someone is enthusiastic about your project, but needs a bit of polish consider mentoring them through their first contribution. mentor someone who’d like to contribute	30	**Category-> contributions-> Practice->Mentoring**
31	Encouraging your community members to work on their own fork can provide the creative outlet they need, without conflicting with your project’s vision	31	**Category-> contributions-> Practice-> Project rules and policies-> allowing project forking by community to avoid conflict with project vision**
32	Require tests and other checks to improve the quality of your code. If you add tests, make sure to explain how they work in your CONTRIBUTING file	32	**Category-> contributions-> Practice-> Project rules and policies-> Add test files and other checks to improve the quality of code**
33	Designating leaders can be as simple as adding their names to your README or a CONTRIBUTORS text file. CONTRIBUTORS or AUTHORS file in your project that lists everyone who’s contributed to your project // **Documentation**	33	**Governance and Leadership -> Practice-> Adding CONTRIBUTORS or AUTHORS file in project to list all contributors**

34	Let people self-organize and volunteer for the roles they're most excited about, rather than assigning them.	34	**Governance and Leadership -> Practice-> Self-organization of contributors by volunteering for the roles they like**
35	Once you've established leadership roles, don't forget to document how people can attain them! **Establish a clear process for how someone can become a maintainer or join a subcommittee in your project, and write it into your GOVERNANCE.md**	35	**Governance and Leadership -> Practice-> Establish a clear process to become a maintainer or join a subcommittee in project, and write it into your GOVERNANCE.md**
36	Under a liberal contribution model, the people who do the most workare recognized as most influential, but this is based on current contributions. Major project decisions are made based on a consensus seeking processwork and not historic	36	**Governance and Leadership -> Practice->Governance model-> the initial author of the project has final say on all major project decisions-> active project contributors (those who demonstrate "merit") make formal decisions-> Major project decisions are based on a consensus seeking process work and not historic**
37	**Post mortem review or after action review** to learn through a collective activity and **to build knowledge based on the experience to improve future practice**. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects	37	**Category-> Knowledge transfer and Sharing->** **Practice-> Post mortem review or after action review to build knowledge based on the experience to improve future practice**
38	A Community of Practice to create a network to collect and exchange information about the knowledge-transfer methods to diffuse knowledge before it is lost	38	**Category-> Knowledge transfer and Sharing-> Practice-> A Community of Practice to create a network to collect and exchange information**
39	Written reports to transfer project relevant knowledge with other contributors	39	**Category-> Knowledge transfer and Sharing-> Practice-> Written reports to transfer project relevant knowledge to contributors**
40	Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices,while **job rotation helps knowledge spread throughout the project or organisation.**	40	**Category-> Uniform knowledge distribution-> Practice-> Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains.**
41	Using story telling to generating, share, and discuss stories for quickly integrating new learning	41	**Category-> Knowledge transfer and Sharing-> Practice-> story telling to generating, share, and discuss stories to integrate new learning**

42	**Experience Based Memory**: Experience packaged and stored in an experience base **built by contributors** based **on** their experiences including **resources such as all experience types, lesson learned, project data, and technology reports**. **Version control, change management, documenting design decisions, and requirements traceability** are software engineering practices that help build project and product memories as an indirect or direct effect of software development. **Relevant technical tools allow for the retention of supporting documents, and competence management and used by software organisations for supporting software engineering practices along with knowledge management. For example, document management tools frequently employed in organisations are Hyperwave, Microsoft SharePoint, Lotus Domino, and Xerox DocuShare. Technology based systems//**	42	**Refer to memo #1, Technology oriented tools->Experience Based Memory-> built by contributors on resources such as all experience types, lesson learned, project data, and technology reports. Version control, change management, documenting design decisions, and requirements traceability that help build project and product memories.**
43	Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing	43	**Category-> Knowledge transfer and Sharing-> Practice-> Training-> include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing**
45	**Rewarding contributors for knowledge sharing and transfer with recognition and appreciation for best answer provided**. Recognition and Reward Structure: In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the organisation. Furthermore, to encourage contribution of knowledge, based on codification and personalisation **a reward system is established for people documenting and sharing knowledge** (Hansen et al. 1999). A reward structure is based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition. Extrinsic motivation is related to monetary incentives (Gammelgaard 2007). Organisations like Xerox, and Hewlett-Packard **reward people for sharing their knowledge** (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003). Using the combination of extrinsic and intrinsic rewards is better (Gammelgaard 2007). In the two types of reward structures, the long lasting one is intrinsic reward structure. As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated. Software engineers willingly share their knowledge. Even though the technology changes very quickly, capturing the gained knowledge still is worth the effort (Rus and Lindvall 2002).	45	**category->intrinsic motivation->subcategory->Rewarding contributors for knowledge sharing and transfer -> recognition and appreciation for best answer provided-> subcategory-> creating a culture to willingly share knowledge with other contributors**

B.4 Categorisation of Practices

Data Component No.	Data Components (now visible as practices)	Memo No:	Categorisation of practices detailed in Memo
1	**Visualisation of Resources tool**- A visualization word cloud has been proposed **to show quickly the level of cooperation of the team in the project** [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors. // **Technology Oriented Tools**	1	Solution that involve technology to support knowledge relavant activites in OSS projects. All such practices that can be used as a rsolution to deal with knowledge loss due to contributor turnover are gathered under one **category "Technology oriented tools"-> Practice-> Visualisation of Resources tool to show quickly the level of cooperation of the team in the project**
2	**Identifying successors with relevant expertise and are knowledgeable on the work of other contributors.** Identification of successors and involving contributors as co-owners with relevant expertise knowledgeable on the work of other contributors. is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10] // **Identification of successors**	2	**Category-> Knowledge transfer and Sharing-> Practice->Identification of successors as co-owners with relevant knowledge on the work of others**
3	**Pair Programming and Shared Code Ownership:** In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. such as pair programming, facilitate knowledge sharing between peers // **Shared code knowledge - pair programming**	3	**Category-> Knowledge transfer and Sharing-> Practice-> Shared code knowledge - pair programming**

4	**Centralisation**: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams. **//Centralisation approach to maintain architectural knowledge and its availability to new developers and teams**	4	Stability of architectural knowledge here means the maintence or updated knowledge in Ericsson during the recurrent movement of people in and out of the product with changing business nees. In survey question rather than using word stable, word maintain is used to avoid confusion of the participants. Because knowledge is always evloving and is never stable. Centralisation approach to maintain architectural knowledge and its availability to new developers and teamsStability of architectural knowledge here means the maintence or updated knowledge in Ericsson during the recurrent movement of people in and out of the product with changing business nees. In survey question rather than using word stable, word maintain is used to avoid confusion of the participants. Because knowledge is always evloving and is never stable. **Category -> Knowledge Centralisation -> Pratcie -> Approach to maintain knowledge centrally and its availability to new developers and teams**
5	**Removal of Knowledge Barriers:** Only a few OSS contributors transit to a higher learning state, due to high learning barriers. Consequently, it can take newcomers up to 60 weeks to become an effective contributor to a OSS project [129]. Knowledge retention in OSS projects can also be improved by the removal of knowledge barriers: namely, lack of **technical experience**, lack of **domain expertise** and lack of **knowledge on project practices** that hinder the contributions [130, 131]. **// Contributions**	5	Removal of knowledge barriers is useful to new contributors on the project to acquire knowledge and to improve the productivity on the project. The knowledge barriers are lower for people who have more experience on the project. So this implies mainly to new commers on the project. **category-> Contributions-> Sub-Category-> removal of knowledge barriers-> Practice -> Improvement in lack of technical experience, lack of domain expertise and lack of knowledge on project practices that hinder the contributions**

6	An insight into the area of expertise members, **a knowledge map or directory** can be used **on the project website** internal knowledge portal, serving as a way **to help knowledge workers familiarize** themselves **with their colleagues knowledge**, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks. // **Technology Oriented tools**	6	memo # 1 **Category -> Adoption os Technology Oriented Tools -> Practice-> Knowledge map or directory on the project website for knowledge workers to familiarize with their colleagues knowledge**
7	**Transactive Memory System**- development within the teams, **based on knowledge location, the usage of the developer mailing list and knowledge credibility // Technology Oriented toolsTransactive Memory System**	7	memo # 1 **Category -> Adoption os Technology Oriented Tools -> Practice-> Transactive Memory System**
8	**Diverse specialisations of Core Contributors**: For an OSS project to survive, a **diversity of core developers** is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant **imbalance between the contribution and the response from the developers' community was noticed.** The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next. **// Balance between the contribution submitted and response from specialised core developers in community. form a "core team" of maintainers, or even** **subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct).**	8	Diversity of core contributors relate to a set of contributors with diverse specialisation. Lack of these contributors delays required feedback on the submitted contribution by non-cores. **The main category is -> Uniform knowledge distribution-> Practice-> diverse specialisation of core contributors-> Practice -> attaining balance between the contribution submitted and response from specialised core contributors in community**

9	**Uniform Knowledge Distribution:** The **communication of the OSS project is directed by the core contributors**. Their **attitudes and involvement in knowledge sharing** were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. **They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities.** The more code changes core developers perform, the more knowledge they provide [124]. However, their least involvement in communication and task changes results into some negative team attitudes. **// Knowledge communication from cores to non-cores**	9	refer to memo #8, The main category is -> **Uniform knowledge distribution**-> Practice-> **Knowledge communication from Cores to non- cores.** **Cores start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities.** Core control the flow of information and knowledge to non cores
10	This kind of disruption in communication in OSS projects can hinder knowledge sharing. Another resolution to the non-uniform distribution of knowledge may be the **proactive assignment of maintenance tasks on the code written by other contributors.** As indicated that contributors who modify codes from other contributors stay longer on the project [28]. This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.	10	refer to memo #8, The main category is -> **Uniform knowledge distribution-> Practice-> proactive assignment of tasks to non-cores->** proactive assignment of maintenance tasks on the code written by other contributors to equally develop skills of non-cores.

11	Gamification: A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133].	11	refer to memo # 45, **Category->Extrinsic motivation-> Practice ->Rewards for knowledge sharing and transfer** **Practice ->Using gamification qualities to evaluate who provides the best answer is the one that gains the most points**
12	Assessing contributors on the meritocracy level for knowledge sharing and transfer similar to the one followed for code contributions on the project	12	refer to memo # 45, **Extrinsic motivation-> Rewards for knowledge sharing and transfer-> Practice-> Assessing meritocracy level of contributors on knowledge sharing and transfer similar to the one followed for code contributions on the project**

13	**Improving Code Review Feedback Time for non-Cores:** Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors [134]. As indicated by a social network analysis of the code review data from eight popular OSS projects, **core developers as compared to peripheral contributors have the benefit of receiving quicker feedback, face shorter review intervals and have a higher code acceptance rate** [135]. Due to the **lack of an established reputation, peripheral developers wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process**. Accordingly, a **delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor** [135]. An improvement to the timings of the review feedbacks in OSS projects to peripherals can result in a **uniform distribution of knowledge**, reduce the effects of turnover, and motivate newcomers to stay for a longer duration.	13	refer to memo # 5. In addition to be a source of negative motivation for new contributors or peripherals, delay in code feedback time is also a barrier to knowledge contributions. **Category-> contributions-> sub category-> removal of knowledge sharing barriers->Practice -> Improving code feedback time for non-cores**
14	Project policy to **invite contributors from all roles to participate in peer review**s to transfer project relevant knowledge and experience	14	**Category-> Uniform Knowledge Distribution-> Practice-> invite contributors from all roles to participate in peer reviews**
15	**Code review checklists also provide team members with clear expectations for each type of review**	15	**refer to memo # 5. Category-> contributions-> sub category-> Project rules and policies->** **Practice -> Code review checklists for team members with clear expectations for each type of review contributions**

16	**Use of a collaborative code review tool** that allows reviewers **to log bugs, discuss them with the author, and approve changes in the code**. Guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options	16	**refer to memo # 5. Category-> contributions-> sub category-> Project rules and policies-> use of collaborative review tool to log bugs, discuss them with the author, and approve changes in the code. Sub category-> Guidelines about project contribution policy with the expected code style, commit format, acceptance and submission process, and available communication options**
17	**Integrators** should be **proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette.** Integrator and contributors should **agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels** (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub) communication is public and accessible, anybody can read past archives to get up to speed and participate	17	**refer to memo # 5. Category-> contributions-> sub category-> Project rules and policies-> Practice -> proactively establishing a professional communication etiquette and minimal communication protocols for mandatory information exchange -> communication is public and accessible, past archives can be read to get up to speed and participate**
18	Good documentation invites people to interact with project open an issue or pull request. Use these interactions as opportunities to move them down the funnel (from users to contributors and to maintainers)	18	**Category-> contributions-> sub category-> Project rules and policies-> Practice ->Good updated documentation for contributors to interact with project and open pull request and progress in project**

19	open-minded about the types of contributions to accept start with a bug report or small fix make easy for casual contributors to contribute. document your process with an access to public	19	**Category-> contributions-> sub category-> Project rules and policies->** **Practice ->Open to all kinds of contributions on the project ->Public access is provided during the documentation of project process**
20	**Transparency about project's roadmap**, the **types of contributions you're looking for**, how contributions are reviewed, or why you made certain decisions	20	**Category-> contributions-> sub category-> Project rules and policies->** **Practice -> Transparency about project's roadmap, the types of contributions required, review process, and rational on decision making**
21	When multiple users running into the same problem, document the answers in the README	21	**Category-> Knowledge transfer Sharing->** **Practice->When multiple users run into the same problem, document the answers in the README**
22	While working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on. //**Encouraging contribution**	22	**Category-> contributions-> sub category-> removal of knowledge sharing barriers->** **Practice -> While working on a substantial update status as work in progress (WIP) for other people to feel involved in the process early on.**

23	to be **responsive when someone files an issue, submits a pull request, or asks a question about your project**. Responding quickly, people will feel they are part of a dialogue, and they'll be more enthusiastic about participating set up notifications in some of these places (such as Stack Overflow, Twitter, or Reddit) so you are alerted when someone mentions your project. if you can't review the request immediately, **acknowledging it early helps increase engagement.**	23	**Category-> contributions-> sub category-> removal of knowledge sharing barriers-> Practice -> Acknowledging or to be responsive when someone files an issue, submits a code, asks questions.**
24	negative people will make other people in your community uncomfortable, a supportive community is the key, resolving conflicts //**Community Health**	24	**Category-> Community Health-> Practice -> supportive community to resolve conflicts**
25	label bugs that are suitable for different types of contributors: for example, "first timers only", "good first issue", or "documentation". These labels make it easy for someone new to your project to quickly scan your issues and get started. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors	25	**Category-> contributions-> sub category-> removal of knowledge sharing barriers-> Practice -> label bugs that are suitable for different types of contributors as opportunities to recruit new contributors**
26	**Sharing project ownership to support user community in absence of project owners. Share ownership of** your **project**, if you need to step away from your project, either on hiatus or permanently, there's no shame in asking someone else to take over for you. Find **support for your users and community while you're away from a project**. If you can't find the support you need, take a break anyway. Be sure to communicate when you're not available, so people aren't confused by your lack of responsiveness.	26	Governance and Leadership practices to continue the project evolving through maintenance when the owner or the main contributor steps away Sharing ownership of the project can be considered a sub category of governance and leadership. **Governance and Leadership -> Practice-> Sharing project ownership to support user community in absence of project owners.**

27	newsletter or write a blog post thanking contributors //**Appreciation**	27	**Intrinsic motivation-> Practice -> appreciation as newsletter or a blog post thanking contributors**
28	Non- restrictive commit access - Give every contributor commit access, this made people more excited to polish their patches	28	**Governance and Leadership -> Practice->Non- restrictive commit access - Give every contributor commit access to polish their patches**
29	Emphasize "consensus seeking" rather than consensus. community members discuss major concerns until they feel they have been adequately heard// **Governance**	29	**Governance and Leadership -> sub-category-> Emphasize "consensus seeking" in community to discuss major concerns until adequately heard**
30	Someone is enthusiastic about your project, but needs a bit of polish consider mentoring them through their first contribution. mentor someone who'd like to contribute	30	**Category-> contributions-> sub category->removal of knowledge barriers -> Practice->Mentoring**
31	Encouraging your community members to work on their own fork can provide the creative outlet they need, without conflicting with your project's vision	31	**Category-> contributions-> sub category-> Project rules and policies-> Practice -> allowing project forking by community to avoid conflict with project vision**
32	Require tests and other checks to improve the quality of your code. If you add tests, make sure to explain how they work in your CONTRIBUTING file	32	**Category-> contributions-> sub category-> Project rules and policies-> Practice -> Add test files and other checks to improve the quality of code**

33	Designating leaders can be as simple as adding their names to your README or a CONTRIBUTORS text file. CONTRIBUTORS or AUTHORS file in your project that lists everyone who's contributed to your project // **Documentation**	33	**Governance and Leadership -> Practice-> Adding CONTRIBUTORS or AUTHORS file in project to list all contributors**
34	Let people self-organize and volunteer for the roles they're most excited about, rather than assigning them.	34	**Governance and Leadership -> Practice-> Self-organization of contributors by volunteering for the roles they like**
35	Once you've established leadership roles, don't forget to document how people can attain them. **Establish a clear process for how someone can become a maintainer or join a subcommittee in your project, and write it into your GOVERNANCE.md**	35	**Governance and Leadership -> Practice-> Establish a clear process to become a maintainer or join a subcommittee in project, and write it into your GOVERNANCE.md**
36	Benevolent Dictator for Life". Under this structure, one person (usually the initial author of the project) has final say on all major project decisions. Under a meritocracy, active project contributors (those who demonstrate "merit") are given a formal decision making role. Under a liberal contribution model, the people who do the most workare recognized as most influential, but this is based on current contributions. Major project decisions are made based on a consensus seeking processwork and not historic	36	**Governance and Leadership -> Practice -> the initial author of the project has final say on all major project decisions-> Practice -> active project contributors (those who demonstrate "merit") make formal decisions** **Practice-> Major project decisions are based on a consensus seeking process work and not historic**

37	**Post mortem review or after action review** to learn through a collective activity and **to build knowledge based on the experience to improve future practice**. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects	37	**Category-> Knowledge transfer and Sharing->** **Practice -> Post mortem review or after action review to build knowledge based on the experience to improve future practice**
38	A Community of Practice to create a network to collect and exchange information about the knowledge-transfer methods to diffuse knowledge before it is lost	38	**Category-> Knowledge transfer and Sharing->** **Practice -> A Community of Practice to create a network to collect and exchange information**
39	Written reports to transfer project relevant knowledge with other contributors	39	**Category-> Knowledge transfer and Sharing-> sub category-> Written reports to transfer project relevant knowledge to contributors**
40	Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices,while **job rotation helps knowledge spread throughout the project or organisation.**	40	**Category-> Uniform knowledge distribution-> Practice-> Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains.**

41	Using story telling to generating, share, and discuss stories for quickly integrating new learning	41	**Category-> Knowledge transfer and Sharing->** **Practice -> story telling to generating, share, and discuss stories to integrate new learning**
42	**Experience Based Memory**: Experience packaged and stored in an experience base **built by contributors** based **on** their experiences including **resources such as all experience types, lesson learned, project data, and technology reports**. **Version control, change management, documenting design decisions, and requirements traceability** are software engineering practices that help build project and product memories as an indirect or direct effect of software development. // **Technology oriented tools**	42	**Refer to memo #1, Category->Technology oriented tools** **Practice->Experience Based Memory-> built by contributors on resources such as all experience types, lesson learned, project data, and technology reports. Version control, change management, documenting design decisions, and requirements traceability that help build project and product memories**
43	Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing	43	**Category-> Knowledge transfer and Sharing->** **Practice -> Training-> include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing**
44	Interviews in an organisation is a knowledge transfer knowledge process used to integrate the knowledge captured into the organisation	44	**Category-> Knowledge transfer and Sharing->** **Practice-> Interviews to integrate the knowledge captured into the organisation**
45	**Rewarding contributors for knowledge sharing and transfer with recognition and appreciation for best answer provided**.	45	**Category->Extrinsic motivation->** **Practice->Rewarding contributors for knowledge sharing and**

	Recognition and Reward Structure: In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the organisation. Furthermore, to encourage contribution of knowledge, based on codification and personalisation **a reward system is established for people documenting and sharing knowledge** (Hansen et al. 1999). A reward structure is based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition. Extrinsic motivation is related to monetary incentives (Gammelgaard 2007). Organisations like Xerox, and Hewlett-Packard **reward people for sharing their knowledge** (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003). Using the combination of extrinsic and intrinsic rewards is better (Gammelgaard 2007). In the two types of reward structures, the long lasting one is intrinsic reward structure. As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated. Software engineers willingly share their knowledge. Even though the technology changes very quickly, capturing the gained knowledge still is worth the effort (Rus and Lindvall 2002).		**transfer ->** **Practice-> recognition and appreciation for best answer provided->** **Practice-> creating a culture to willingly share knowledge with other contributors**

B.5 PKR Practices and Categories - Revisiting Change and Renaming

Proactive Knowledge Retention Practices Category	Proactive Knowledge Retention Practices
Adoption of Technology Oriented Tools (5)	**Visualisation of Resources tool** to show quickly the level of cooperation of the team in the project
	Knowledge map or directory on the project website for knowledge workers to familiarize with their colleagues knowledge
	Transactive Memory System based on knowledge location, the usage of the developer mailing list and knowledge credibility
	Experienced based memory: built by contributors on resources such as all experience types, lesson learned, project data, and technology reports. Version control, change management, documenting design decisions, and requirements traceability, help build project and product memories
	Use of collaborative review tool to log bugs, discuss them with the author, and approve changes in the code**Use of collaborative review tool** to log bugs, discuss them with the author, and approve changes in the code
Knowledge transfer and Sharing (9)	**Identification of successors as co-owners** with relevant knowledge on the work of others
	When multiple users run into the same problem, document the answers in the README
	Post mortem review or after action review to build knowledge based on the experience to improve future practice
	A Community of Practice to create a network to collect and exchange information
	Written reports to transfer project relevant knowledge to contributors
	Story telling to generating, share, and discuss stories to integrate new learning
	Training Include combination of formal classroom training, eLearning, video or computerbased training, on the job training, coaching, and shadowing
	Shared code knowledge using **pair programming**

	Interviews to integrate the knowledge captured into the organisation
Knowledge Centralisation (1)	**Approach to maintain knowledge centrally** and its availability to new developers and teams
Encouraging Knowledge Contributions (16)	**Removal of knowledge sharing barriers:**
	1. **Mentoring**
	2. **Improving code feedback time for non-cores**
	3. **Improvement** in lack of technical experience, lack of domain expertise and lack of knowledge on project practices that hinder the contributions
	4. **Updating work status** to work in progress (WIP),while working on a substantial for other people to feel involved in the process early on.
	5. **Acknowledging or to be responsive** when someone files an issue, submits a code, asks questions
	6. **Label bugs** that are suitable for different types of contributors as opportunities to recruit new contributors
	Standarising project rules and policy:
	1. **Code review checklists** for team members with clear expectations for each type of review contributions
	2. **Guidelines on project contribution policy** with the expected code style, commit format, acceptance and submission process, and available communication options
	3. **Proactively establishing communication** etiquette and minimal communication protocols for mandatory information exchange
	4. **Communication is public and accessible**, past archives can be read to get up to speed and participate

Encouraging Knowledge Contributions (16)	5. **Good updated documentation** for contributors to interact with project and open pull request and progress in project
	6. **Openness** to all kinds of contributions on the project
	7. **Public access** is provided during the documentation of project process
	8. **Transparency** about project's roadmap, the types of contributions required, review process, and rational on decision making.
	9. **Add test files** and other checks to improve the quality of code
	10. **Allowing project forking** by community to avoid conflict with project vision
Uniform knowledge distribution (6)	**Diverse specialisation of core** contributors
	Attaining balance between the contribution submitted and response from specialised core contributors in community
	Knowledge communication from Cores to noncores
	Proactive assignment of tasks to noncores
	invite contributors from all roles to participate in peer reviews
	Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains.
Motivation (6)	**Intrinsic Motivation:**
	Appreciation written as newsletter or a blog post thanking contributors
	Recognition and appreciation for best answer provided
	Creating a culture to willingly share knowledge with other contributors
	Extrinsic motivation:
	Rewarding contributor for knowledge sharing and transfer

	Using **gamification** qualities to evaluate who provides the best answer is the one that gains the most points
	Ascending meritocracy level of contributors on knowledge sharing and transfer activities similar to the one followed for code contributions on the project
Community health (1)	**Resolving conflicts** with the help of supportive community
Governance and Leadership (8)	**Sharing project ownership** to support user community in absence of project owners
	Emphasize "consensus seeking" in community to discuss major concerns until adequately heard
	Non restrictive commit access-Give every contributor commit access to polish their patches
	Adding CONTRIBUTORS or AUTHORS file in project to list all contributors
	Encouraging self-organization of contributors by volunteering for the roles they like
	Establish a clear process to become a maintainer or join a subcommittee in project, and write it into GOVERNANCE.md
	The initial author of the project has final say on all major project decisions
	Major project decisions are based on a consensus seeking process work and not historic
	Active project contributors (those who demonstrate "merit") make formal decisions

B.6 PKR Practices - First Review

Practice	Outcome	Memos / Proactive Knowledge Retention Practice Category
Adopting real-time visualisation of resources	Timely deployment of the resources on need basis with the help of the technology. Relocating knowledge workers based on technology. Timely deployment of the resources on need basis with the help of the technology.	Resources are managed in case of turnover using visualisation of resources. Tracking resources on the project and on need basis provide resources. Knowledge Resource Management
Keeping track of successor/ co-owner/ knowledgeable person/ with relevant expertise on the work of others	Reduces the risk of file/ code/ module abandonment.	Knowledge Resource Management
Pair programming and shared code ownership.	Knowledge sharing and transfer among contributors relating to specific code.	Knowledge Transfer and Sharing
Centralisation approach to update architectural knowledge and its availability to contributors and teams Instead of maintain using the word update	Knowledge centralisation ensures accessibility of updated knowledge to contributors and teams and secures quality of the developed system.	Knowledge Maintenance, ensures updated knowledge accessible to all contributors
Training contributors in OSS projects	Training can be also given by mentoring contributors for quality contributions. Removal of knowledge barriers: namely, lack of technical experience, lack of domain expertise and lack of knowledge on project practices that hinder the contributions.	Removal of Knowledge Barriers
Developing Transactive Memory Systems/Knowledge map/ Knowledge portals/ based on knowledge location, using developers mailing lists and knowledge credibility.	KMS will direct the contributors to the knowledge location or the knowledgeable person, their expertise, and familarise themselves with member's knowledge.	Knowledge Resource Management
Diversifying specialisation of core contributors	Attaining balance between the contribution submitted and response from specialised core developers in community. Form a “core team” of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct).	Removal of Knowledge Barriers
Knowledge communication from cores to non-cores	This practice leads to uniform knowledge distribution on the project, wehre the communication is directed by the cores. Cores are the initiater of the project and most knowledgeable ones. Knowledge sharing, knowledge transfer and uniform knowledge distribution.	Removal of Knowledge Barriers

Proactive assignment of varying tasks to a contributor	Helps in uniform knowledge distribution, knowledge transfer, knowledge sharing, equally develop skills of non-cores.	Knowledge Transfer and Sharing
Rewarding knowledge provider's by recognition	Increasing the engagment of knowledge providers, knowledge sharing and transfer. Encouragment and motivation for knowledge sharing	Extrinsic Motivation
Progress in meritocracy based on knowledge sharing and transfer activities	Encouraging OSS community to share knowledge with other contributors	Extrinsic Motivation
Improving code review feedback time for non-cores	Facilitates knowledge exchange and serves as positive motivation for non-cores.	Removal of Knowledge Barriers
Project policy to invite contributors from all roles for peer reviews	Contributes to uniform knowledge distribution, tranfers project relevant knowledge and experiences	Uniform Knowledge Distribution
Maintaining code review checklist	Faciliates in a succesful quality contribution. Information for knowledge seekers	Quality Knowledge Contributions
Adopting use of collaborative tool for code review	Improves code quality and knowledge sharing	Quality Knowledge Contributions
Establishing communication protocol between integrator and contributors	knowledge exchange and encouraging contributions from new participants by following on the communication from archives	Knowledge Exchange
Frequent updation of project documents.	improves understandibility of the project and makes it easier to contribute to the project	Removal of Knowledge Barriers
Acceptability to all types of knowledge contributions	Allows for casual contributors to contribute	Openness towards knowledge contributions
Public access during documentation of project process	provides visibility of project processes to contributors. Project specific knowledge	Transparency to project specific knowledge
Documenting project rules, policies, contribution review process and decisions	Transparency to rules and policy gives an understanding of project and expectations from the contributors	Project specific knowledge
Listing solutions to problems faced by multiple users	Saves time and efforts to resolve an issue if the solution is already stated. Helps contributor to resolve issues	Project specific knowledge
Labeling ongoing work as work in progress	Improves the involvement of contributors and creates visibility of an update ongoing.	Knowledge Exchange
Timely response to issues submitted or queries on the project	Early acknowledgment helps increase engagement on the project	Knowledge Sharing Barriers
Building a supportive community for conflict resolution	has positive effect on community health and healthy environment for knowledge sharing and transfer	Knowledge Transfer and Sharing
Labeling bugs to suit contributors with different level of expertise	Acts as an opportunity to recruit newknowledge contributors	Removal of Knowledge Barriers
Sharing project ownership	Extended support for users and community in the absence of the main owners.	Knowledge Maintenance

Appreciating contributors more often	Intrinsic motivation for contributors and feel included in the project	Extrinsic Motivation
Non- restrictive commit access	Give every contributor commit access, this made people more excited to polish their patches	Knowledge Maintenance
Consensus seeking approach	Liberal structure for community involvement and earning their trust that their concerns are heard. Effective knowledge sharing because liberty to share their views.	Knowledge Exchange
Mentoring	successful and quality contributions from contributors. Overcoming knowledge barriers for new comers	Removal of Knowledge Barriers
Project Forking	avoiding conflicts on the project	Project specific knowledge
Presence of Test Files including checks to test code	Knowledge artefact improves the quality of contributor code	Project specific knowledge
Listing contributors/ authors on Project	Information for newcomers/ new contributors who to ask for help on the project.	Project specific knowledge
Allowing self-organization on project roles	Improve the overall productivity and work environment on the project. Facilitate knowledge sharing and transfer	Knowledge Transfer and Sharing
Eliciting process to become maintainer	knowledge on the project specific process. Expected to improve involvement from contributor on the project.	Project specific knowledge
Post-mortem reviews	Build a knowledge base consulted by knowledge seekers to learn from experience	Knowledge Transfer and Sharing
Establishing a Community of practice network to collect and exchange knowledge	knowledge-transfer methods to diffuse knowledge before it is lost	Knowledge Transfer and Sharing
Report writing	Written reports to transfer project relevant knowledge to contributors	Project specific knowledge
Job rotation	knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains.	knowledge Exchange and transfer
Story telling	to share, and discuss stories for quickly integrating new learning. Knowledge transfer and sharing	Knowledge Transfer and Sharing

Experience based memory – built by contributors on resources: lesson learned/ project data/Version control/ change management/ documenting design decisions/ and requirements traceability	Knowledge retention practice to build project memories	Project specific knowledge
Training/ e-Learning/ video/ computer-based training/ on-the-job training/ coaching/ and shadowing	Knowledge transfer, knowledge sharing	Knowledge Transfer and Sharing
Interviews	interviews to capture knowledge and then transfer it	Knowledge Transfer and Sharing
Creating culture to share knowledge altruistically	Awareness among contributors to share knowledge. Intrinsic motivation for altruism	Intrinsic Motivation

B.7 PKR Practices - Second Review

Proactive Knowledge Retention Practice Category	KPR Practice	Memos
Communication	**Knowledge communication from cores to non-cores**	Knowledge communication between cores and non-cores is an important factor for knowledge trasnfer and sharing leading to uniform knowledge distribution
	Improving code review feedback time for non-cores	Responsivness in communicating code review feedback for non-cores in OSS project facilitates knowledge sharing and transferf rom sores to non-cores. This practice is included in communication category.
	Establishing communication protocol between integrator and contributors	Communication protocol elicited is included under the category communication. Communication is important to the progress and success of the project and contributors need to understand what the protocol is.
	Timely response to submitted project issues or queries	To encourage engagement on the project
	Consensus seeking approach	This practice is more related to an approach adopted to reach out to contributors to find their views, which is the mode of communication. This practice can also be included governance and leadership category. But is more appropriate to be considered in communication category because it demonstrates a communication mode for the community to collaborate.
Contributor Motivation	**Rewarding knowledge provider's by recognition**	Reward is associated with extrinsic motivation of a contributor. The category contributor motivation relates to encouraging knowledge providers by rewarding them.
	Appreciating contributors more often	Appreciating contributors more often by writing in blogs or newsletters about their positive efforts to keep the project on going.

Core Development Practice	**Pair programming and shared code ownership.**	Pair programming helps to share and transfer knowledge among contributors. It can be one of the core fundemental practice in OSS for knowledge transfer and sharing and for having a constant knowledge transfer and sharing mechanism in place.
	Diversifying specialisation of core contributors	The core contributors should have diverse specialisation. In order to attain diverse specialisation in OSS projects, this should be a core development practice. Here core means fudemental not core development team in OSS projects.
	Project policy to invite contributors from all roles for peer reviews	Peer review is a collaborative activity and inviting contributors from different roles can lead to knowledge transfer and sharing. This is included as the core devlopment practice since peer review is one of the key activities in OSS development.
	Labeling bugs for contributors with varying Skills & Expertise	Labeling a bug to suit the level of contributor expertise will benfit OSS project to recruit more contributors and get them started with simpler tasks suitable to their knowledge and experience.
	Presence of Test Files and checks to test code	Test files to test code can be one of the core deveopment practices and directly impact the quality of the code. It helps to test code for any defects and facilitates the contributor to write quality code.
	Listing solutions to problems faced by multiple users	During the development, OSS can face many issue where some are common for many users. It is a usful practice and therefore should be placced in core development practices.
	Labeling ongoing work as work in progress	Labelling on going work as work in process attracts more contributors interested in the ongoing updates. Furthermore, more expertise in the OSS development is pooled in by such labelling activity. It can be a piece of information that is communicated to contributors.

	Non- restrictive commit access	In OSS project not every contributor has right to commit the code. The code review policies describe the process for commit access. Generally code commit access is given to only selected contributors. If contributors are allowed to commit their code to a repository where it doesn't effect main branch but at the same time contributor code is reviewed by other contributors. This expedites the learning and correction process. This practices falls well under the category of Core Development Practices.
Environment/ Ecosystem/ Culture	**Building a supportive community for conflict resolution**	A OSS community with an ability to resolve conflicts will last longer and with time evolve and more productive. Building a community that is supportive is related to Environment/ Ecosystem/ Culture.
	Creating culture to share knowledge altruistically	Creating awareness among contributors and involving altruistic philosophy that is the main reason to initiate an OSS project. Knowledge sharing can be introduced as a culture in OSS to stablise ecosystem and selfless genture of doing good to society.
	Public access during project process documentation	Public access provides visibility to ongoing project documentation. This can be related to the communicating chnages in the documentation on the fly. But it is more of an adoption of a culture or environment where documentation are given public access to be viewed by the contributors of the project. it is also effective practice for the ecosystem in OSS community.
	Acceptability to all types of knowledge contributions **Acceptability to all types of knowledge contributions**	OSS project governing teams and other contributors should adopt a culture to be acceptable to all kinds of contributions including small ones. This kind of open culture encourages contributors who are just casual contributors but their efforts are recognised and accepted.

	Encourage Project Forking	A OSS community with an ability to resolve conflicts will last longer and with time evolve and more productive. A project community and governance that encourage project forking to avoid conflict with the project's visions and goals. this practice should be acceptable in OS communities and iss related to Environment/ Ecosystem/ Culture.
Governance and Leadership	**Keeping track of successor/ co-owner/ knowledgeable person/ with relevant expertise on the work of others**	The team leaders would be keeping track and identifying people who are knowledgeable on the work of others. It can also be the culture/ Environmet on the project to keep track on the work of other contributors. A teamleader would have more resources to track people with similar area of knowledge while for an individual contributor it will be only known to him or her that who is working on code simialr to hers. OSS projects will benefit more if sucessors/ owners/ based on similar work history are tracked by the team leaders.
	Centralising knowledge from all sources **Centralising knowledge from all sources**	This can be related to culture of OSS projects. Having a culture solely would not fulfill the need of contributors when it comes to accessibility. The knowledge centralisation and updation has to be monitored by team leaders and its acessibility to contributors and teams. Also team leads or people governing OSS project will be more knowledgeable to update the documents at the central location.
	Developing Transactive Memory Systems / Knowledge map / Knowledge portals	KMS system development has to be directed by the leaders of the projects. It helps in knowledge transfer and sharing. Once the KMS are in place, it can be maintained by the contributors.

Proactive assignment of varying tasks to non-cores	The skill set of core contributors is better than non-cores. Cores make 80% of the knowledge contributions in OSS projects. In order to uniformly distribute knowledge from cores to non-cores, the team leads can assign different tasks to non-cores proactively. This falls under government and leadership because the team leads can control the extent of knowledge that is shared with non-cores. proactive assignemnt of different tasks to non-cores can be governed by leaders in OSS projects.
Maintaining code review checklist	This practice is related to Governance and Leadership since the policy of code review are given and monitored by the team leaders
Adopting use of collaborative tool for code review	Adopting a tool for code review is a decision to be made by the leadership on the project.
Sharing project ownership	Sharing project ownership is related to helping community find support in the absence of the main leader and to keep the maintenance on the project ongoing. This practice is categorised under Governance and Leadership
Eliciting process to become maintainer	The policy to become a maintainer should be elicited by the team leaders on the project. It is engaging for the contributors to see what can be outcome of their efforts and their personal progress on the project. This also serves as a motivation to do well to escalate to the highest level of recognition on the project.
Document project rules, policies,review process and decisions	All these documents are to be provided by the project leaders. There can be changes made later to the rules and policies but the responsibility of documenting this rules and policies mainly lies with the leadership. Also for a person joining a new project, provision of these documents enables them to evaluate on the basis of the elaborated rules and policies

	Adopting real-time visualisation of resources	Project leaders, owners or team leaders in OSS may relocate resources on projects based on the visualisation of resources when there are less resources than required. Decisions to use a technology tool that depicts the real-time visualisation of resources is mainly upon the main leaders of the project
Knowledge Transfer and Sharing	**Post-mortem reviews for learning**	Post mortem reviews are conducted in project at different stages. It is learning based on the experience and is an effective way to tansfer and share knowledge. This practice is categirised under knowledg transfer and sharing. Post mortem review can be effective way for knowledge transfer for OSS contributors while learning from experiences discussed on the past projects
	Establishing Community of practice network	Community of practices focuses on learning from experience of others with a focus on establishing a network. It helps to transfer and share knowledge. The practice is categorised under knowledg transfer and sharing
	Written reports for project relevant knowledge	Reporting writing is considered another effective way to share and transfer project relevant knowledge.
	Job rotation to acquire different skills	Job rotation on a project that allows a contributor to work under different roles in a project to acquire different skills. In OSS project contributors at a time can working on multiple roles, this practices ensures that contributors atleast work on all roles in a Project. Job rotation leads to effective knowledge transfer and sharing.
	Story telling to integrate learning	Contributors in OSS project are encouraged to share knowledge through story telling, which invloves sharing and transfering their learning. The story telling is a knowledge transfer and sharing practice.

	Building memories on project documents and data	Projects memories are made with the data avilable from the project in the form of lesson learned, project data, version control. This practice can be categorised under knowledge transfer and sharing.
	e-Learning/ video/ computer-based training/ on-the-job training/ coaching/ and shadowing	Some more practices related to knowledge transfer and sharing. Some are of digital nature, other relate to knowledge transfer while on job.
	Knowledge capture and transfer by interviews	Interviews with contributors highlight knowledge specific to the project. This is knowledge cature and then transfering it in the form of writing or digitally. Interviews are categorised under knowledge transfer and sharing.
Removal of Knowledge Barriers	**Training contributors**	Training of the contributor is appropriate to overcome the knowledge barriers in OSS projects. Removal of knowledge barriers: namely, lack of technical experience, lack of domain expertise and lack of knowledge on project practices that hinder the contributions.
	Mentoring Contributors on Project	Mentoring contributors enthusiastic to work on the project for a head start and submit quality contributions. This preactice is more useful to overcome knowledge barriers face by contributors mainly who are new to the project.
	Frequent updation of project documents.	This practice is useful for the removal of knowledge barriers. It is the responsibility of the contributor working on the project to ensure that documents are up to date. In case of problem or clarification required on the process they should bring to the attention of team leaders.

B.8 PKR Practices - Third Review

Proactive Knowledge Retention Practice Category	KPR Practice	KPR Practice Description Rev 1 Rev 2 Rev 3	Memos
Communication	**Establish communication mechanisms from cores to non-cores contributors**	Knowledge communication from cores to non-cores Communicate high level ideas, suggestions, information, comments, instructions and answers to team members. Enagage in more code changes as core developers and through this share more knowledge with other team members. Knowledge communication from cores to non-cores	Knowledge communication between cores and non-cores is an important factor for knowledge trasnfer and sharing leading to uniform knowledge distribution The communication of the OSS project is directed by the core contributors. Their attitudes and involvement in knowledge sharing were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. They start the project and provide high levels of ideas, suggestions, information, comments, instructions and answers to their teams, and are the centre of their project's knowledge activities. The more code changes core developers perform, the more knowledge they provide [124]. However, their least involvement in communication and task changes results into some negative team attitudes. Knowledge communication between cores and non-cores is an important factor for knowledge trasnfer and sharing leading to uniform knowledge distribution
	Advocating manimum code review feedback time for non-core contributors	Improving code review feedback time for non-cores. Set a maximum time limit to respond to submitted code review. Both cores and non-cores should be treated equally and without any dintinction.Improving code review feedback time for non-cores.	Responsivness in communicating code review feedback for non-cores in OSS project facilitates knowledge sharing and transfer from cores to non-cores. This practice is included in communication category. This practice can also be categorised as "Core Development Practices". I kept in communication responsiveness and acknowledgement is related to communication issues rather than development practice. Due to the lack of an established reputation, peripheral developers wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process. Accordingly, a delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor [135]. An improvement to the timings of the review feedbacks in OSS projects to peripherals can result in a uniform distribution of knowledge, reduce the effects of turnover, and motivate newcomers to stay for a longer duration.

Clarification of communication protocol/ mechanisms between integrator and contributors	**Integrators should be proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette. Integrator and contributors should agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub) communication is public and accessible, anybody can read past archives to get up to speed and participate.**	Communication protocol elicited is included under the category communication. Communication is important to the progress and success of the project and contributors need to understand what the protocol is. Integrators should be proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette. Integrator and contributors should agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real- time communication channels (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub) communication is public and accessible, anybody can read past archives to get up to speed and participateCommunication protocol elicited is included under the category communication. Communication is important to the progress and success of the project and contributors need to understand what the protocol is.
Establishing response time parameters for project queries or issues	**to be responsive when someone files an issue, submits a pull request, or asks a question about your project. Responding quickly, people will feel they are part of a dialogue, and they'll be more enthusiastic about participating set up notifications in some of these places (such as Stack Overflow, Twitter, or Reddit) so you are alerted when someone mentions your project. if you can't review the request immediately, acknowledging it early helps increase engagement.**	To encourage engagement on the project
Actively eliciting views of project community on major concerns	**Consensus seeking approach. Emphasize "consensus seeking" rather than consensus. Community members discuss major concerns until they feel they have been adequately heard. While sensible at first glance, voting emphasizes getting to an "answer," rather than listening to and addressing each other's concerns.** **As a project maintainer, it's important other contributors on the project know that their opinion is heard and someone in leading position is listening. Making other people feel heard, and committing to resolving their concerns, goes a long way to diffuse sensitive situations. Make sure that everybody feels heard and that all information has been made public before moving toward a resolution.**	This practice is more related to an approach adopted to reach out to contributors to find their views, which is like initiating the communication. This practice can also be included governance and leadership category. But is more appropriate to be considered in communication category because it demonstrates a communication mode for the community to collaborate.

		Under a liberal contribution model, generally people who do the most work are recognized as most influential, and it is based on current contributions. Major project decisions are made based on a consensus seeking process.	
Contributor Motivation	**Establishing a knowledge recognition program**	**Rewarding knowledge provider's by recognition** **Organisations like Xerox, and Hewlett-Packard reward people for sharing their knowledge (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003).** **To encourage contribution of knowledge, based on codification (formally documenting) and personalisation (sharing knowledge through socialisation) a reward system is established for people documenting and sharing knowledge (Hansen et al. 1999)Rewarding knowledge provider's by recognition**	Reward is associated with extrinsic motivation of a contributor. The category contributor motivation relates to encouraging knowledge providers by rewarding them.
	Offering progress as motivation to knowledge sharing and transfer	**Gamification: A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities**	Contributors progress to become a core contributor in the project based on meritocracy and mainly on the number of the code contributions they submit. The practice suggests to utilise the concept of extrinsic motivation and have an additional criteria of assessing a contributor based on knowledge sharing and transfer activities on the project. Contributors progress to become a core contributor in the project based on meritocracy and mainly on the number of the code contributions they submit. The practice suggests to utilise the concept of extrinsic motivation and have an additional criteria of assessing a contributor based on knowledge sharing and transfer activities on the project. Contributors progress to become a core

		are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133]. Here the number of points gained by the contributor can serve as a extrinsic motivation to gain recognition and earn a repute in OSS community. At the same time gamification identifies people who are more knowledgeable and active based on the point given to them. vation to gain recognition and earn a repute in OSS community.	contributor in the project based on meritocracy and mainly on the number of the code contributions they submit. The practice suggests to utilise the concept of extrinsic motivation and have an additional criteria of assessing a contributor based on knowledge sharing and transfer activities on the project.
	To engage in active appreciation for contributors	**Appreciating contributors more often. newsletter or write a blog post thanking contributors**	Appreciating contributors more often by writing in blogs or newsletters about their positive efforts to keep the project on going. Appreciation is an extrinsic motivation.
Core Development Practice	**Pair programming and shared code ownership.**	**Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. such as pair programming, facilitate knowledge sharing between peers. Remote pair programming is not just for the corporate sector; it can also be quite successful in open-source projects. Remote pair programming can be as simple and efficient as it is in person programming. Using a text editor where a workspace and some form of video chatting is shared. Basically one does same things as in person with a little bit more articulation of thoughts since your body language is not visible.** **//Expert–novice**	Pair programming helps to share and transfer knowledge among contributors. It can be one of the core fundemental practice in OSS for knowledge transfer and sharing and for having a constant knowledge transfer and sharing mechanism in place.

		Expert–novice pairing creates many opportunities for the expert to mentor the novice. This pairing can also introduce new ideas, as the novice is more likely to question established practices. The expert, now required to explain established practices, is also more likely to question them. However, in this pairing, an intimidated novice may passively "watch the master" and hesitate to participate meaningfully. Also, some experts may not have the patience needed to allow constructive novice participation. (ref: Williams, L. & Kessler, R. (2003). Pair Programming Illuminated. Boston: Addison-Wesley Professional.)Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. such as pair programming, facilitate knowledge sharing between peers.	
	Diversifying specialisation of core contributors	**Diverse specialisations of Core Contributors: For an OSS project to survive, a diversity of core developers is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant imbalance between the contribution and the response from the developers' community was noticed. The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next. Balance between the contribution submitted and response from specialised core developers in community.** **Form a "core team" of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct).**	The core contributors should have diverse specialisation. In order to attain diverse specialisation in OSS projects, this should be a core development practice. Here core means fudemental not core development team in OSS projects.

Project policy to invite contributors from all roles for peer reviews	**Core team proactivly invites contributors irrespective of their role on the project encourages them to participate in peer reviews.**	Peer review is a collaborative activity and inviting contributors from different roles can lead to knowledge transfer and sharing. This is included as the core devlopment practice since peer review is one of the key activities in OSS development.
Labeling bugs for contributors with varying Skills & Expertise	**label bugs that are suitable for different types of contributors: for example, "first timers only", "good first issue", or "documentation". These labels make it easy for someone new to your project to quickly scan your issues and get started. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors**	Labeling a bug to suit the level of contributor expertise will benfit OSS project to recruit more contributors and get them started with simpler tasks suitable to their knowledge and experience.
Presence of Test Files and checks to test code	**Require tests and other checks to improve the quality of the code. If tests are added, explain how they work in a file designated for this purpose e.g. CONTRIBUTING.**	Test files to test code can be one of the core deveopment practices and directly impact the quality of the code. It helps to test code for any defects and facilitates the contributor to write quality code.
Listing solutions to problems faced by multiple users	**When multiple users running into the same problem, document the answers in the some file such as README.**	During the development, OSS can face many issue where some are common for many users. It is a usful practice and therefore should be placced in core development practices.
Labeling ongoing work as work in progress	**While working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on. Encouraging contribution while working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on.**	Labelling on going work as work in process attracts more contributors interested in the ongoing updates. Furthermore, more expertise in the OSS development is pooled in by such labelling activity. It can be a piece of information that is communicated to contributors.

	Non- restrictive commit access	**Give every contributor commit access, to allow people to be more excited to polish their patches even can help in finding new maintainers for projects that had not been worked on in a while.** **Give every contributor commit access, to allow people to be more excited to polish their patches even can help in finding new maintainers for projects that had not been worked on in a while.**	In OSS project not every contributor has right to commit the code. The code review policies describe the process for commit access. Generally code commit access is given to only selected contributors. If contributors are allowed to commit their code to a repository where it doesn't effect main branch but at the same time contributor code is reviewed byy other contributors. This expedites the learning and correction process. This practices falls well under the category of Core Development Practices.
Environment/ Ecosystem/ Culture	**Building a supportive community for conflict resolution**	**Any popular project will inevitably attract people who harm, rather than help, your community. They may start unnecessary debates, quibble over trivial features, or bully others.** **Negative people will make other people in community uncomfortable. Project leaders should adopt a zero-tolerance policy towards these types of people. If left unchecked, negative people will make other people in community uncomfortable. They may even leave.** **Adopting a code of conduct builds a supportive community is the key to resolving conflicts. "A code of conduct is a document that establishes expectations for behavior for project's participants. Adopting, and enforcing, a code of conduct can help create a positive social atmosphere for project community."**	A OSS community with an ability to resolve conflicts will last longer and with time evolve and more productive. Building a community that is supportive is related to Environment/ Ecosystem/ Culture.
	Listing contributors/ authors on Project	**Designating leaders can be as simple as adding their names to README or a CONTRIBUTORS text file.** **CONTRIBUTORS or AUTHORS file in the project that lists everyone who's contributed to the project.** **For a bigger project, if there is a website, create a team page or list project leaders there. For example, Postgres has a comprehensive team page with short profiles for each contributor.**	This practice can be categorised under governance and leadership, since they are the leaders of the project. But it is more suitable to categorise it under Environment/ Ecosystem/ Culture because then everyone can add their name to the list and it is more openly accessible.

Allowing self-organization on project roles	**Let people self-organize and volunteer for the roles they're most excited about and interested in, rather than assigning them.**	self organisation is related to letting people decide what role they want on the project. It can be categorised under governance and leadership. It can also be categorised under Environment/ Ecosystem due to reason that contributors interact in OSS ecosystem and decide on the role they want and communicate it effectively. Under governance and leadership the self-organisation is portrayed as decision making restricted only to team leaders.
Creating culture to share knowledge altruistically	**As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated. Software engineers willingly share their knowledge.**	Creating awareness among contributors and involving altruistic philosophy that is the main reason to initiate an OSS project. Knowledge sharing can be introduced as a culture in OSS to stablise ecosystem and selfless genture of doing good to society.
Public access during project process documentation	**Document process with an access to public. Communication is public and accessible, anybody can read past archives to get up to speed by going though mailing lists, blogs and participate.** **Public communication can be as simple as directing people to open an issue instead of emailing you directly or commenting on your blog. You could also set up a mailing list, or create a Twitter account, Slack, or IRC channel for people to talk about your project.**	Public access provides visibility to ongoing project documentation. This can be related to the communicating chnages in the documentation on the fly. But it is more of an adoption of a culture or environment where documentation are given public access to be viewed by the contributors of the project. it is also effective practice for the ecosystem in OSS community.
Acceptability to all types of contributions	**Open-minded about the types of contributions to accept start with a bug report or small fix making it easier for casual contributors to contribute**	OSS project governing teams and other contributors should adopt a culture to be acceptable to all kinds of contributions including small ones. This kind of open culture encourages contributors who are just casual contributors but their efforts are recognised and accepted.

	Encourage Project Forking	**Encouraging your community members to work on their own fork can provide the creative outlet they need and exyend the current work. Users can fultill their legitimate needs as a user of the code by making changes required without conflicting with project's vision. A document can guide contributors on forking the project.**	A OSS community with an ability to resolve conflicts will last longer and with time evolve and be more productive. A project community and governance that encourage project forking to avoid conflict with the project's visions and goals. this practice should be acceptable in OS communities and issues related to Environment/ Ecosystem/ Culture.
Governance and Leadership	**Frequent updation of project documents**	**Centralisation: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams.**	This can be related to culture of OSS projects. Having a culture solely would not fulfill the need of contributors when it comes to accessibility. The knowledge centralisation and updation has to be monitored by team leaders and its acessibility to contributors and teams. Also team leads or people governing OSS project will be more knowledgeable to update the documents at the central location.
	Keeping track of successor/ co-owner/ knowledgeable person/ with relevant expertise on the work of others	**Identification of successors and involving contributors as co-owners with relevant expertise knowledgeable on the work of other contributors is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10].** **Developing Transactive Memory Systems / Knowledge map / Knowledge portals (OSS Literature)** **The team leaders would be able to keep the track and** **identifying people who are knowledgeable on the work of others.**	It can also be the culture/ Environmet on the project to keep track on the work of other contributors. A teamleader would have more resources to track people with similar area of knowledge while for an individual contributor it will be only known to him or her that who is working on code simialr to hers. OSS projects will benefit more if sucessors/ owners/ based on similar work history are tracked by the team leaders. KMS system development has to be directed by the leaders of the projects. It helps in knowledge transfer and sharing. Once the KMS are in place, it can be maintained by the contributors.

	Transactive Memory System-development within the teams, based on knowledge location, the usage of the developer mailing list and knowledge credibility. An insight into the area of expertise members, a knowledge map or directory can be used on the project website internal knowledge portal, serving as a way to help knowledge workers familiarize themselves with their colleagues knowledge, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks	
Proactive assignment of varying tasks to non-cores	**As indicated that contributors who modify codes from other contributors stay longer on the project [28]. This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.**	The skill set of core contributors is better than non-cores. Cores make 80% of the knowledge contributions in OSS projects. In order to uniformly distribute knowledge from cores to non-cores, the team leads can assign different tasks to non-cores proactively. This falls under government and leadership because the team leads can control the extent of knowledge that is shared with non-cores. proactive assignemnt of different tasks to non-cores can be governed by leaders in OSS projects.
Maintaining code review checklist	**Code review checklists also provide team members with clear expectations for each type of review.**	This practice is related to Governance and Leadership since the policy of code review are given and monitored by the team leaders
Adopting use of collaborative tool for code review	**Use of a collaborative code review tool that allows reviewers to log bugs, discuss them with the author, and approve changes in the code guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options**	Adopting a tool for code review is a decision to be made by the leadership on the project.

	Sharing project ownership	**Share ownership of the project, if the maintainer need to step away from project, either on hiatus or permanently, request other contributors to take over. Find support for your users and community while you're away from a project.Be sure to communicate you are not available, so people are not confused by your lack of responsiveness.**	Sharing project ownership is related to helping community find support in the absence of the main leader and to keep the maintenance on the project ongoing. This practice is categorised under Governance and Leadership
	Eliciting process to become maintainer	**Document the process on the project to become maintainer. Use these interactions happening through documents as opportunities to help contributors move down the funnel (from users to contributors and to maintainers)** **Document the process on the project to become maintainer.**	The policy to become a maintainer should be elicited by the team leaders on the project. It is engaging for the contributors to see what can be outcome of their efforts and their personal progress on the project. This also serves as a motivation to do well to escalate to the highest level of recognition on the project.
	Document project rules, policies,review process and decisions	**Transparency about project's roadmap, the types of contributions required for the project, how contributions are reviewed, or why certain decisions are made on the project** **Transparency about project's roadmap, the types of contributions required for the project, how contributions are reviewed, or why certain decisions are made on the project**	All these documents are to be provided by the project leaders. There can be changes made later to the rules and policies but the responsibility of documenting this rules and policies mainly lies with the leadership. Also for a person joining a new project, provision of these documents enables them to evaluate on the basis of the elaborated rules and policies
	Adopting real-time visualisation of resources	**A visualization word cloud has been proposed to show quickly the level of cooperation of the team in the project [21]. A large variety of data collection is without human intervention and rendered as a wordle. Intensity of colour and size of the letter in a wordle indicate a need for resources. A Visualization word cloud does not cause overhead on the productivity of the contributors.**	Project leaders, owners or team leaders in OSS may relocate resources on projects based on the visualisation of resources when there are less resources than required. Decisions to use a technology tool that depicts the real-time visualisation of resources is mainly upon the main leaders of the project

Knowledge transfer and sharing	**Post-mortem reviews for learning**	**Post mortem review or after action review to learn through a collective activity and to build knowledge based on the experience to improve future practice. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects.** **In Apple a similar approach is used where results are published based on a project survey, collecting useful information on project, a debriefing meeting, and a 'project history day'. Microsoft invests considerable efforts into writing 'Postmortem reports'. The reports contain details on what has worked well for the last project, what has not worked well and teams that need improvement for the next project.**	Post mortem reviews are conducted in project at different stages. It is learning based on the experience and is an effective way to tansfer and share knowledge. This practice is categirised under knowledg transfer and sharing. Post mortem review can be effective way for knowledge transfer for OSS contributors while learning from experiences discussed on the past projects.
	Establishing Community of practice network	**A Community of Practice to create a network of experienced contributors to collect and exchange knowledge. Siemens and BMW to resolve the challenges of losing expertise through retirement and attrition found a cross-company community network. Intel, Infineon Technologies, and Winterthur Insurance Switzerland joined the "Leaving Experts Community of Practice". Members of the community must be experienced in dealing with the problem of losing expertise and be able to contribute valuable lessons to member companies.**	Community of practices focuses on learning from experience of others with a focus on establishing a network. It helps to transfer and share knowledge. The practice is categorised under knowledg transfer and sharing

	Job rotation to acquire different skills	**Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices, such as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation.**	Job rotation on a project that allows a contributor to work under different roles in a project to acquire different skills. In OSS project contributors at a time can working on multiple roles, this practices ensures that contributors atleast work on all roles in a Project. Job rotation leads to effective knowledge transfer and sharing.
	Story telling to integrate learning	**Using story telling to generating, share, and discuss stories for quickly integrating new learning. NASA, the World Bank, IBM, made productive use of storytelling**	Contributors in OSS project are encouraged to share knowledge through story telling, which invloves sharing and transfering their learning. The story telling is a knowledge transfer and sharing practice.
	Building memories on project documents and data	**Experience Based Memory: Experience packaged and stored in an experience base built by contributors based on their experiences including resources such as all experience types, lesson learned, project data, and technology reports. Version control, change management, documenting design decisions, and requirements traceability are software engineering practices that help build project and product memories as an indirect or direct effect of software development.**	Projects memories are made with the data avilable from the project in the form of lesson learned, project data, version control. This practice can be categorised under knowledge transfer and sharing.
	Training based on e-Learning/ video/ computer-based/ on-the-job/ coaching/ and shadowing	**Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing (De Long and Davenport 2003).**	Some more practices related to knowledge transfer and sharing. Some are of digital nature, other relate to knowledge transfer while on job.
	Knowledge capture and transfer by interviews	**Interviews in an organisation is a knowledge transfer knowledge process used to integrate the knowledge captured into the organisation**	Interviews with contributors highlight knowledge specific to the project. This is knowledge cature and then transfering it in the form of writing or digitally. Interviews are categorised under knowledge transfer and sharing.

Removal of Knowledge Barriers	**Training contributors**	**Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing**	Training of the contributor is appropriate to overcome the knowledge barriers in OSS projects. Removal of knowledge barriers: namely, lack of technical experience, lack of domain expertise and lack of knowledge on project practices that hinder the contributions.
	Mentoring contributors on project	**Someone is enthusiastic about the project, but needs a bit of polish consider mentoring them through their first contribution.** **Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors and mentor someone who would like to contribute**	Mentoring contributors enthusiastic to work on the project for a head start and submit quality contributions. This practice is more useful to overcome knowledge barriers face by contributors mainly who are new to the project.
	Frequent updation of project documents	**keep project documentation up-to-date**	This practice is useful for the removal of knowledge barriers. It is the responsibility of the contributor working on the project to ensure that documents are up to date. In case of problem or clarification required on the process they should bring to the attention of team leaders.

B.9 PKR Practices - Fourth Review

Proactive Knowledge Retention Practice Category (32)	KPR Practice	KPR Practice Description Rev 1 Rev 2 Rev 3 Rev 4	Memo 4
Communication (5)	Establishing communication mechanisms from cores to non-cores contributors	Advocate active communication on matters such as high level ideas, suggestions, information, comments, instructions, answers to team members and also includes engaging with software code. The more changes made to the code, the more knowledge core contributors share with others specially non-core contributors. This practice should result in community of practice to create a network of experienced contributors to collect and exchange knowledge. For example, Siemens and BMW to resolve the challenges of losing expertise through retirement and attrition found a cross-company community network. Intel, Infineon Technologies, and Winterthur Insurance Switzerland joined the “Leaving Experts Community of Practice”. Members of the community must be experienced in dealing with the problem of losing expertise and be able to contribute valuable lessons to member companies.	Changed the practice name from "Establishing Community of practice network". Rephrased the practice description. Added practice "Establishing Community of practice network" under knowledge sharing and transfer under this practice as an example.
	Advocating maximum code review feedback time for all contributors	Improving code review feedback time for non-cores. Set a maximum time limit to respond to submitted code review by cores and non-cores alike.Improving code review feedback time for non-cores.	Rephrased the practice description

	Clarification of communication protocol/ mechanisms between integrators and contributors	Integrators should be proactive by establishing a professional communication etiquette, and reactive, by following discussions and intervening in cases where discussion diverges from the etiquette. Integrator and contributors should agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub) communication is public and accessible, anybody can read past archives to get up to speed and participate. Using story telling to generating, share, and discuss stories for quickly integrating new learning. NASA, the World Bank, IBM, made productive use of storytelling. Interviews in an organisation is a knowledge transfer knowledge process used to integrate the knowledge captured into the organisation.	"Story telling to integrate learning" from the category knowledge transfer and sharing is added as an example to this practice. Contributors in OSS project are encouraged to share knowledge through story telling, which invloves sharing and transfering their learning. The story telling is a knowledge transfer and sharing practice. "Knowledge capture and transfer by interviews" practice is removed from knowledge sharing and transfer category and moved under category communication. Interviews with contributors highlight knowledge specific to the project. This is knowledge cature and then transfering it in the form of writing or digitally. Interviews are categorised under knowledge transfer and sharing.
	Establishing response time parameters for project queries or issues	Responding effectively when someone files an issue, submits a pull request, or asks a question about the project. Responding quickly makes project community feel that they are part of a dialogue, and builds their enthusiasm to participate. In case, the request cannot be reviewed immediately, acknowledging it early helps increase engagement among project community. For example, setting up notifications in some of places like Stack Overflow, Twitter, or Reddit issue an alert when someone mentions your project.	Changes made to the descritpion of the practice.

	Actively eliciting views of project community on major concerns	As a project maintainer, it's important other contributors on the project know that their opinion is heard and someone in leading position is listening. Making other people feel heard, and committing to resolving their concerns, goes a long way to diffuse sensitive situations. Make sure that everybody feels heard and that all information has been made public before moving toward a resolution. Under a liberal contribution model, generally people who do the most work are recognized as most influential, and it is based on current contributions. Major project decisions are made based on a consensus seeking process. Example by elicitng views of project contributors , one should emphasise consensus seeking approach. Emphasize "consensus seeking" rather than consensus. Community members discuss major concerns until they feel they have been adequately heard. While sensible at first glance, voting emphasizes getting to an "answer," rather than listening to and addressing each other's concerns.	No changes made to this practice or its description This practice is more related to an approach adopted to reach out to contributors to find their views, which is like initiating the communication. This practice can also be included governance and leadership category. But is more appropriate to be considered in communication category because it demonstrates a communication mode for the community to collaborate.No changes made to this practice or its description
	Promoting documentation of solutions to problems commonly faced by multiple users	When multiple users running into the same problem, document the answers in the some file such as README. Not a description….	Changed the name of the practice from "Listing solutions to problems faced by multiple users" to the current name for understanding and calrification.
Contributor Motivation (3)	Establishing a formal knowledge contribution recognition program	Rewarding knowledge provider's by recognition Organisations like Xerox, and Hewlett-Packard reward people for sharing their knowledge (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003). To encourage contribution of knowledge, based on codification (formally documenting) and personalisation (sharing knowledge through socialisation) a reward system is established for people documenting and sharing knowledge (Hansen et al. 1999).	No changes made to this practice or its description

	To reward active knowledge contributors with project seniority	Description..... Gamification: A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133]. Here the number of points gained by the contributor can serve as a extrinsic motivation to gain recognition and earn a repute in OSS community. At the same time gamification identifies people who are more knowledgeable and active based on the point given to them.	Changed the name of the practice from "Offering progress as incentives as a motivation to knowledge sharing and transfer" to the current one.
	To promote an active culture of appreciation for contributors	Appreciating contributors more often. newsletter or write a blog post thanking contributors more frequently Appreciating contributors more often. newsletter or write a blog post thanking contributors more frequently Appreciating contributors more often. newsletter or write a blog post thanking contributors more frequently	Changed the name of the practice from "To engage in active appreciation for contributors" to the current one for clarity.

Core Development Practice (8)	Encouraging pair programming and a culture of shared code ownership	Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. such as pair programming, facilitate knowledge sharing between peers.
		
		// continuity sentence
		
		Remote pair programming is not just for the corporate sector; it can also be quite successful in open-source projects. Remote pair programming can be as simple and efficient as it is in person programming. Using a text editor where a workspace and some form of video chatting is shared. Basically one does same things as in person with a little bit more articulation of thoughts since you body language is not visible.
		
		//Expert–novice
		Expert–novice pairing creates many opportunities for the expert to mentor the novice. This pairing can also introduce new ideas, as the novice is more likely to question established practices. The expert, now required to explain established practices, is also more likely to question them. However, in this pairing, an intimidated novice may passively "watch the master" and hesitate to participate meaningfully. Also, some experts may not have the patience needed to allow constructive novice participation. (ref: Williams, L. & Kessler, R. (2003). Pair Programming Illuminated. Boston: Addison-Wesley Professional.)Pair Programming and Shared Code Ownership: In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. such as pair programming, facilitate knowledge sharing between peers.
			Some changes to the name of the practice for clarification

	Project policy to invite contributors from all roles for peer reviews	Core team proactivly invites contributors irrespective of their role on the project encourages them to participate in peer reviews.	No changes made to this practice or its description.
	Labeling bugs for contributors for different levels of contributors	label bugs that are suitable for different types of contributors: for example, “first timers only”, “good first issue”, or “documentation”. These labels make it easy for someone new to your project to quickly scan your issues and get started. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors	Changed the name of the practice from "Labeling bugs for contributors with varying Skills & Expertise" to the current name for understanding and calrification.

Ensuring the presence of test files and associated testing artefacts	Require tests and other checks to improve the quality of the code. If tests are added, explain how they work in a file designated for this purpose e.g. CONTRIBUTING.	Changed the name of the practice from "Presence of Test Files and checks to test code" to the current name for understanding and calrification.
Substantial new items of work are labeled as work in progress	While working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on. Encouraging contribution while working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on.	Changed the name of the practice from "Labeling ongoing work as work in progress" to the current name for understanding and calrification.
Introduce non-restrictive commit access to contributors where appropriate	Give every contributor commit access, to allow people to be more excited to polish their patches even can help in finding new maintainers for projects that had not been worked on in a while.	Changed the name of the practice from "Introduce non-restrictive commit access to contributors where appropriate" to the current name for understanding and calrification.
Establishing explicit code review guidelines	Use of a collaborative code review tool that allows reviewers to log bugs, discuss them with the author, and approve changes in the code guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options. Code review checklists also provide team members with clear expectations for each type of review.	The name of the practice is changed from "Maintaining code review checklist" to the current name. This practice is moved to core development practice from governance and leadership category. The practice after the internal review is thought to be more suitable under core development practices because it provides insights to code review guidelines. Another practice"Adopting use of collaborative tool for code review" under governance and leadership is merged with this practice. and details of the included practice are under the practice descritption.

	Release post-mortem reviews	Post mortem review or after action review to learn through a collective activity and to build knowledge based on the experience to improve future practice. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects. In Apple a similar approach is used where results are published based on a project survey, collecting useful information on project, a debriefing meeting, and a 'project history day'. Microsoft invests considerable efforts into writing 'Postmortem reports'. The reports contain details on what has worked well for the last project, what has not worked well and teams that need improvement for the next project.Post mortem review or after action review to learn through a collective activity and to build knowledge based on the experience to improve future practice. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects.	The name of the practice is changed from "Post-mortem reviews for learning". The practice of post-mortem can be more effective in OSS projects at the time of release. The practice is also removed from the knowledge sharing and transfer category since it is agreed to be more related to Core dvelopment practices category. The practice can help in retaining knowledge from the experience and lesson learned on the project.
vironment/ Ecosystem / Culture (7)	Building a supportive community for conflict resolution	Any popular project will inevitably attract people who harm, rather than help, your community. They may start unnecessary debates, quibble over trivial features, or bully others. Negative people will make other people in community uncomfortable. Project leaders should adopt a zero-tolerance policy towards these types of people. If left unchecked, negative people will make other people in community uncomfortable. They may even leave. Adopting a code of conduct builds a supportive community is the key to resolving conflicts. "A code of conduct is a document that establishes	No changes are made to this practice.

Env		expectations for behavior for project's participants. Adopting, and enforcing, a code of conduct can help create a positive social atmosphere for project community." Any popular project will inevitably attract people who harm, rather than help, your community. They may start unnecessary debates, quibble over trivial features, or bully others. Negative people will make other people in community uncomfortable. Project leaders should adopt a zero-tolerance policy towards these types of people. If left unchecked, negative people will make other people in community uncomfortable. They may even leave.	
	Explicity identify contributors on the project	Designating leaders can be as simple as adding their names to README or a CONTRIBUTORS text file. CONTRIBUTORS or AUTHORS file in the project that lists everyone who's contributed to the project. For a bigger project, if there is a website, create a team page or list project leaders there. For example, Postgres has a comprehensive team page with short profiles for each contributor.	Changed the name of the practice from "Listing contributors/ authors on Project" to the current name for understanding and calrification.An example of how changing names makes the practice more useful for the project contributors and how effectively they can use it to benefit the project.
	Allowing self-organization of project roles	Let people self-organize and volunteer for the roles they're most excited about and interested in, rather than assigning them.	No changes were made to this practice and its description.
	Creating a culture to share knowledge altruistically	As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated.	No changes made to this practice and its description.
	Making project documentation publically accessible	Project document should be openly accessible an access to public. Communication is public and accessible, anybody can read past archives to get up to speed by going though mailing lists, blogs and participate. Public communication can be as simple as directing people to open an issue instead of emailing you directly or commenting on your blog. You could also set up a mailing list, or create a Twitter account, Slack, or IRC channel for people to talk about your project.Project document should be openly accessible an access to public. Communication is public and accessible, anybody can read past archives to get up to speed by going though mailing lists, blogs and participate.	Changed the name of the practice from "Public access during project process documentation" to the current name for understanding and calrification.

	Fostering an openminded culture towards diverse types of contributions	Open-minded about the types of contributions to accept start with a bug report or small fix making it easier for casual contributors to contribute	Changed the name of the practice from "Acceptability to all types of contributions" to the current name for understanding and calrification.
	To encourage mentorship of new comers	Someone is enthusiastic about the project, but needs a bit of polish consider mentoring them through their first contribution. Resist fixing easy (non-critical) bugs. Instead, use them as opportunities to recruit new contributors and mentor someone who would like to contribute Someone is enthusiastic about the project, but needs a bit of polish consider mentoring them through their first contribution.	changed the name of the pratice from "Mentoring contributors on project" and moved it from the category "removal of knowledge barriers" to "culture/ ecosystem/ environment"
	Encourage Project Forking	Encouraging your community members to work on their own fork can provide the creative outlet they need and extend the current work. Users can fultill their legitimate needs as a user of the code by making changes required without conflicting with project's vision. A document can guide contributors on forking the project.	After the review it was the decided to exclude this practice because it does not benefit the project directly. It serves as an encouragement for the contributors to take a different direction to the project. Additionally, this practice does not serve as a candidate effective knowledge retention practices, which is the ultimate goal of this activity.
Governance and Leadership (8)	Frequent updation of project documents	example of practice Centralisation: Governance structures in Ericsson is argued to be similar to OSS, and a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams. Actively maintain the document is another example	"actively maintain the documents" moved from "Removal of knowledge barrier " category under "Governance and leadership". "actively maintain the documents" moved from "Removal of knowledge barrier " category under "Governance and leadership".

Enabling a strategy of successor identification	Identification of successors and involving contributors as co-owners with relevant expertise knowledgeable on the work of other contributors is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10]. For example, adopting real-time visualisation of resources: Project leaders, owners or team leaders in OSS may relocate resources on projects based on the visualisation of resources when there are less resources than required. Decisions to use a technology tool that depicts the real-time visualisation of resources is mainly upon the main leaders of the project Developing Transactive Memory Systems / Knowledge map / Knowledge portals (OSS Literature) The team leaders would be able to keep the track and identifying people who are knowledgeable on the work of others. Transactive Memory System- development within the teams, based on knowledge location, the usage of the developer mailing list and knowledge credibility. An insight into the area of expertise members, a knowledge map or directory can be used on the project website internal knowledge portal, serving as a way to help knowledge workers familiarize themselves with their colleagues knowledge, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks	The name of the practice is changed from "Keeping track of successor/ co-owner/ knowledgeable person/ with relevant expertise on the work of others". The indicated name provides a better insight into what exactly this knowledge retention practice does. Add "Adopting real-time visualisation of resources" tot he current as an example rather than practice listed under from governance and leadership under this practice. Project leaders, owners or team leaders in OSS may relocate resources on projects based on the visualisation of resources when there are less resources than required. Decisions to use a technology tool that depicts the real-time visualisation of resources is mainly upon the main leaders of the project
Proactive assignment of varying tasks to non-cores Proactive assignment of varying tasks to non-cores	As indicated that contributors who modify code from other contributors stay longer on the project [28]. outcome: This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.	No changes were made to this practice and its description.

Advocating diversification of core contributors specialisation Advocating diversification of core contributors specialisation	Diverse specialisations of Core Contributors: For an OSS project to survive, a diversity of core developers is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant imbalance between the contribution and the response from the developers' community was noticed. The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next. Balance between the contribution submitted and response from specialised core developers in community. Form a "core team" of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct). Job rotation for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices, such as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation.	This practice is moved from "Core Development Practice" category under "Governance and Leadership". The practice above this one also refelcts on the doversification of contributors specifically non-cores. While this practice focus on cores. The main objective of both practices is same only targetted to different set of contributors. Name of the practice is also changed for a better understanding. "Job rotation to acquire different skills" Is moved from knowledge sharing and transfer category as as example to practice "Advocating diversification of core contributors specialisation " This practice is moved from "Core Development Practice" category under "Governance and Leadership". The practice above this one also refelcts on the doversification of contributors specifically non-cores. While this practice focus on cores. The main objective of both practices is same only targetted to different set of contributors. Name of the practice is also changed for a better understanding. "Job rotation to acquire different skills" Is moved from knowledge sharing and transfer category as as example to practice "Advocating diversification of core contributors specialisation "
Distributed project leadership	Share ownership of the project, if the maintainer need to step away from project, either on hiatus or permanently, request other contributors to take over. Find support for your users and community while you're away from a project. Be sure to communicate you are not available, so people are not confused by your lack of responsiveness.	The name of the practice is changed from "sharing project ownership" to the current name.

	Explicit process for role progression	Document the process on the project to become a maintainer. Use these interactions happening through documents as opportunities to help contributors move down the funnel (from users to contributors and to maintainers) Document the process on the project to become a maintainer. Use these interactions happening through documents as opportunities to help contributors move down the funnel (from users to contributors and to maintainers)	The name of the practice is changed from "Eliciting process to become maintainer" to the current name. Role progression is an important incentive for the contributors to have quality contributions.
	Document project rules and policies	Transparency about project's roadmap, the types of contributions required for the project, how contributions are reviewed, or why certain decisions are made on the project	changed names of the practice from "Document project rules, policies,review process and decisions".
	Training contributors	Training is another knowledge transfer practice found to include some combination of formal classroom training, eLearning, video or computer-based training, on-the-job training, coaching, and shadowing (De Long and Davenport 2003). Training based on e-Learning/ video/ computer-based/ on-the-job/ coaching/ and shadowing // "Establishing mechnisms for training" as new governance and leadership	Created a new practice here and as an example moved practice "Training based on e-Learning/ video/ computer-based/ on-the-job/ coaching/ and shadowing" moved from category knowledge transfer and sharing and now placed under governance and leadership.

B.10 PKR Practices - Fifth Review

Proactive Knowledge Retention Practice Category (32)	KPR Practice	KPR Practice Description Rev 1 Rev 2 Rev 3 Rev 4 Rev 5	Memo 5
Communication (6)	Establishing communication mechanisms from cores to non-cores contributors	The communication of the OSS project is directed by the core contributors. Their attitudes and involvement in knowledge sharing were linked to the demands of their wider project teams [124]. The core contributors bring high levels of skills and cognitive characteristics to their project teams. However, their least involvement in communication and task changes results into some negative team attitudes. Knowledge communication between cores and non-cores is an important factor for knowledge transfer and sharing leading to uniform knowledge distribution This practice advocates active communication on matters such as high level ideas, suggestions, information, comments, instructions, answers to other contributors and also includes engaging with software code. The more changes made to the code, the more knowledge core contributors share with others specially non-core contributors. This practice should result in 'community of practice' to create a network of experienced contributors to collect and exchange knowledge with non-cores. For example, Siemens and BMW to resolve the challenges of losing expertise through retirement and attrition found a cross-company community network. Intel, Infineon Technologies, and Winterthur Insurance Switzerland joined the “Leaving Experts Community of Practice”. Members of the community must be experienced in dealing with the problem of losing expertise and be able to contribute valuable lessons to member companies. The above practice can be adapted for OSS project by creating a community of practice by involving experienced cores, who contribute towards sharing valuable lessons learned and experience on the project to non-cores. Another example of communication mechanism from cores to non-cores in OSS projects can be of using story telling to generate, share, and discuss stories for quickly integrating new learning. NASA, the World Bank, IBM, made productive use of storytelling. Third example for the said practice can be of conducting interviews, which is a knowledge transfer process used to integrate the knowledge captured into the organisation. In OSS projects, interview can serve to transfer knowledge from cores to non-cores.	**Revised the description of the practice for readibility and understanding.**

Advocating maximum limit of code review feedback time for all contributors	Due to the lack of an established reputation, peripheral developers or non-cores wait 2 to 19 times (or 12 to 96 hours) longer than core developers, to complete the review process. Accordingly, a delay in receiving feedback on reviews may negatively motivate a peripheral or new contributor [135]. An improvement to the timings of the review feedbacks in OSS projects of peripherals or non-cores can result in a uniform distribution of knowledge, reduce the effects of turnover, and motivate newcomers to stay for a longer duration. This practice is included in communication category but can be categorised under "Core Development Practices" or under 'Motivation'. In this practice the emphasis is on providing timely feedback and minimising delay, which is more related to communication category rather than 'core development' or 'motivation' category. An example of overcoming delay to core review feedback time, is to set a maximum time limit to respond to a submitted code review by cores and non-cores alike.	**Revised the description of the practice for readability and understanding.**
Clarification of communication protocol/ mechanisms between integrators and contributors	Integrator is a person who merges a branch (a subset of the same project) into the main branch of the project. Integrators should be proactive by establishing a professional communication etiquette, and reactive by following discussions and intervening in cases where discussion diverges from the etiquette. For instance, integrator and contributors should agree on minimal communication protocols that increase each other's awareness and rendezvous points for mandatory information exchange and use real-time communication channels (e.g., IRC or its evolved counterpart GITTER, which is better integrated in GitHub) communication is public and accessible, anybody can read past archives to get up to speed and participate.	**Revised the description of the practice for readability and understanding.**
Establishing response time parameters for project queries or issues	This practice focuses on the timely response to queries or issues on the project helps to engage contributors and makes them feel involved in the project. Furthermore, interest of contributors looking forward to get involved in the project will be heightened when they receive timely response to their queries resulting in new contributors on the project. Responding effectively when someone files an issue, submits a pull request, or asks a question about the project. Responding quickly makes project community feel that they are part of a dialogue, and builds their enthusiasm to participate. In case, the request cannot be reviewed immediately, acknowledging it early helps increase engagement among project community. For example, setting up notifications in some of places like Stack Overflow, Twitter, or Reddit issue an alert when someone mentions your project.	**Revised the description of the practice for readability and understanding.**

	Actively eliciting views of project community on major concerns	Maintainer has commit rights on the project specifically the main branch. Contributor who wants to make a modification ('contributor') creates a branch and makes the changes to that branch, then sends a request to the project 'maintainers' (people who have commit rights to the project). As a project maintainer, it's important for other contributors on the project know that their opinion is heard and someone in leading position is listening. Making other people feel heard, and committing to resolving their concerns, goes a long way to diffuse sensitive situations. Make sure that everybody feels heard and that all information has been made public before moving toward a resolution. Under a liberal contribution model, generally people who do the most work are recognized as most influential, and it is based on current contributions. Major project decisions are made based on a consensus seeking process. This practice focuses more on approach adopted to reach out to contributors to find out their views on major concerns through open discussion. This practice can also be included under governance and leadership category. One of the responsibility of governance in project is to voice the opinion of people. This practice adheres more to achieving communication objectives on the project rather than governing the project. It is more appropriate to consider this practice under communication category of proactive knowledge retention practices. Example by eliciting views of project contributors, one should emphasis consensus seeking approach. Emphasize "consensus seeking" rather than consensus. Community members discuss major concerns until they feel they have been adequately heard. While sensible at first glance, voting emphasizes getting to an "answer," rather than listening to and addressing each other's concerns.	**Revised the description of the practice for readability and understanding.**
	Promoting documentation of solutions to problems commonly faced by multiple users	OSS users can face many issues while using the project. This practice promotes the documentation of solutions to problems experienced by users. This practice belongs in the communication category while demonstrating the use of documentation as an effective mode to communicate knowledge to the contributors of the project. The issues that are commonly faced by many users are suggested to be recorded in writing and document is given access to all contributors on the project. Example of implementing this practice can be of when multiple users run into the same problem, document the answers in the some file such as README and provide its access to contributors.	**Revised the description of the practice for readability and understanding.**

Contributor Motivation (3)	Establishing a formal knowledge contribution recognition program	In order to encourage employees to participate in KR activities, a recognition and reward structure can be incorporated in the core processes of the organisation. A reward structure is based on using either intrinsic motivators or extrinsic motivators. Intrinsic motivator includes acts that make the job more satisfying such as praise and recognition. Extrinsic motivation is related to monetary incentives Reward is associated with extrinsic motivation of a contributor. The category contributor motivation relates to encouraging knowledge providers by rewarding them. Organisations like Xerox, and Hewlett-Packard reward people for sharing their knowledge (Rus and Lindvall 2002). Reward system is not only associated with the sharing of existing knowledge but also with the external knowledge from outsiders. Managers are rewarded in organisations for learning from their competitors, which are source of external knowledge (Menon and Pfeffer 2003). To encourage contribution of knowledge, based on codification (formally documenting) and personalisation (sharing knowledge through socialisation) a reward system is established for people documenting and sharing knowledge (Hansen et al. 1999)Rewarding knowledge provider's by recognition	**Revised the description of the practice for readability and understanding.**
	To reward active knowledge contributors with project seniority	Contributors progress to become a core contributor in the project based on meritocracy and mainly on the number of the code contributions they submit. The practice suggests to utilise the concept of extrinsic motivation and have an additional criteria of assessing a contributor based on knowledge sharing and transfer activities on the project. Active knowledge sharing can be seen as a mechanism to contribute towards contributor's seniority. A gamified environment has important implications for knowledge management in software engineering [50] and OSS projects. As observed, Q&A gamification increased the engagement of knowledge providers and the quickness of response. This finding suggests that Q&A site designers should consider gamification elements to increase contributor engagement, which indirectly can help to raise the popularity of their sites. For example, gamification features in Stack Overflow's guarantee that a question will be replied to by enthusiastic experts within minutes of being posted [133]. On Stack Overflow, a crowd approach is used where participants contribute knowledge independently of each other and gamification qualities are used to evaluate who provides the best answer is the one that gains the most points [133]. Knowledge is curated in gamification other than being developed as is the case with mailing lists. Curation is a mechanism to provide a tool for keeping the channel clean of what seems to be unnecessary information [133]. Here the number of points gained by the contributor can serve as a extrinsic motivation to gain recognition and earn a repute in OSS community. At the same time gamification identifies people who are more knowledgeable and active based on the point given to them.	**Revised the description of the practice for readability and understanding.**

	To promote an active culture of appreciation for contributors	Appreciation of contributors is an extrinsic type of motivation. Appreciating contributors more often through activities such as newsletters and blog posts while thanking contributors motivates them to stay on the project. This practice encourages the adoption of such culture where objective is to motivate contributors by appreciation of their efforts in the project.	**Revised the description of the practice for readability and understanding.**
Core Development Practice (8)	Encouraging pair programming and a culture of shared code ownership	In order to mitigate the effects of turnover on the ecosystem, the usage of techniques such as pair programming and shared code ownership are suggested [95]. Pair programming facilitates knowledge sharing between peers. This practice is considered under the category of core development practice represents proactive approach towards knowledge retention in OSS projects. It further can facilitate effective knowledge sharing in the OSS projects. Examples of pair programming discussed here are remote pair programming and expert novice pairing. Remote pair programming is not just for the corporate sector; it can also be quite successful in open-source projects. Remote pair programming can be as simple and efficient as it is in person programming. Using a text editor where a workspace and some form of video chatting is shared. Basically one does same things as in person with a little bit more articulation of thoughts since you body language is not visible. Expert–novice pairing creates many opportunities for the expert to mentor the novice. This pairing can also introduce new ideas, as the novice is more likely to question established practices. The expert, now required to explain established practices, is also more likely to question them. However, in this pairing, an intimidated novice may passively "watch the master" and hesitate to participate meaningfully. Also, some experts may not have the patience needed to allow constructive novice participation. (ref: Williams, L. & Kessler, R. (2003). Pair Programming Illuminated. Boston: Addison-Wesley Professional.)	**Revised the description of the practice for readability and understanding.**
	Project policy to invite contributors from all roles for peer reviews	Peer reviews are conducted asynchronously in OSS projects to empower experts who provide feedback to code contributors [134]. Peer review is a collaborative activity and inviting contributors from different roles can lead to uniform knowledge distribution from cores to non-cores. This is included as the core development practice since peer review is one of the key development activity in OSS development. An example of this practice can be that core team proactively invites contributors irrespective of their role on the project and encourage them to participate in peer reviews.	**Revised the description of the practice for readability and understanding.**

Labelling bugs for contributors for different levels of contributors	Labelling a bug to suit the level of contributor expertise will benefit OSS project to recruit more contributors and get them started with tasks according to their knowledge and expertise. Labelling bugs helps in identifying types of tasks and facilitate their resolution based on the suitability of contributor. This practice is included under the 'core development practice', because it benefits the development of the OSS project. For example, “first timers only”, “good first issue”, or “documentation”. These labels make it easy for someone new on the project to quickly scan issues on the project and get started. Similarly, resisting to fix easy (non-critical) bugs provides opportunities to recruit new contributors	**Revised the description of the practice for readability and understanding.**
Ensuring the presence of test files and associated testing artefacts	Test files and testing artefacts help in testing code and can be one of the core development practices to directly impact the quality of the code. Test files include tests and other checks to improve the quality of the code. For example, if tests are added for the project, explanation should be provided on how they work in a file specifically designated for this purpose e.g. CONTRIBUTING.	**Revised the description of the practice for readability and understanding.**
Substantial new items of work are labelled as work in progress	Labelling on going work as work in progress attracts more contributors interested in the ongoing updates. Furthermore, more expertise in the OSS development is pooled in by such labelling activity. It can be a piece of information on current work that is communicated to contributors. This practice encourages and invites more contribution while working on a substantial update and therefore is included in 'core development practice'. The practice can be applied, while working on a substantial project update, through a pull request and marking it as a work in progress (WIP). That way, other people can feel involved in the process early on.	**Revised the description of the practice for readability and understanding.**
Introduce non-restrictive commit access to contributors where appropriate	In OSS project not every contributor has right to commit the code. The code review policies describe the process for commit access. Generally code commit access is given only to selected contributors. If contributors are allowed to commit their code to a repository where it doesn't effect main branch but at the same time contributor code is reviewed by other contributors. This expedites the learning and correction process. This practices falls under the category of Core Development Practices. Practically the practice can be applied by giving every contributor commit access, to allow her to be more excited to polish her patches. Further benefit of applying this practice can be that it helps in finding new maintainers for projects that had not been worked on in a while.	**Revised the description of the practice for readability and understanding.**
Establishing explicit code review guidelines	Code review guidelines are effective for code reviews and corrections. This practice belongs to 'Core Development Practice' category. Code review checklists can be a useful way to provide team members with clear expectations for each type of review. The use of a collaborative code review tool is recommended that allows reviewers to log bugs, discuss them with the author, and approve changes in the code guidelines should at least provide details about the expected code style, commit format, pull request process, and available communication options.	**Revised the description of the practice for readability and understanding.**

	Release post-mortem reviews	Post mortem review or after action review to learn through a collective activity and to build knowledge based on the experience to improve future practice. In postmortem, learning takes place through socialisation, and when individuals share experiences, tacit knowledge is externalised. Knowledge is shared from individual level to organisational level. Postmortems are an attempt to codify knowledge from projects, where the main output is the report. It can be seen as a systematic mechanism of capturing, storing, interpreting and distributing relevant experience from projects. In Apple a similar approach is used where results are published based on a project survey, collecting useful information on project, a debriefing meeting, and a 'project history day'. Microsoft invests considerable efforts into writing 'Postmortem reports'. The reports contain details on what has worked well for the last project, what has not worked well and teams that need improvement for the next project.	**Revised the description of the practice for readability and understanding.**
Environment/ Ecosystem/ Culture (7)	Building a supportive community for conflict resolution	Any popular project will inevitably attract people who harm, rather than help project's community. They may start unnecessary debates, quibble over trivial features, or bully others. Negative people will make other people in community uncomfortable. Project leaders should adopt a zero-tolerance policy towards these types of people. If left unchecked, negative people will make other people in community uncomfortable. They may even leave. A OSS community with an ability to resolve conflicts will last longer and with time evolve and more productive. Building a community that is supportive is related to Environment/ Ecosystem/ Culture. For example, adopting a code of conduct builds a supportive community is the key to resolving conflicts. "A code of conduct is a document that establishes expectations for behaviour for project's participants. Adopting, and enforcing, a code of conduct can help create a positive social atmosphere for project community."	**Revised the description of the practice for readability and understanding.**
	Explicitly identify contributors on the project	Explicit identification of contributors who are involved with the project can be helpful for reaching out to them for queries, solutions, and support. This practice is relevant to environment/ ecosystem/ culture of the OSS project community. Example of identifying the contributors can be that designating leaders simply as add their names to README or a CONTRIBUTORS text file. CONTRIBUTORS or AUTHORS file in the project that lists everyone who's contributed to the project. For a bigger project, if there is a website, creating a team page or list project leaders would help. For instance, Postgres has a comprehensive team page with short profiles for each contributor.	**Revised the description of the practice for readability and understanding.**

Allowing self-organization of project roles	Self organisation is related to letting people decide what role they want on the project. It can be categorised under governance and leadership. It can also be categorised under Environment/ Ecosystem due to reason that contributors interact in OSS ecosystem and decide on the role they want and communicate it to others effectively. Under governance and leadership the self-organisation is portrayed as decision making restricted only to team leaders. Alternatively, this practice under environment/ ecosystem/ culture category reflects more freedom to allow self-organisation among contributors. For example, letting people self-organize and volunteer for the roles they're most excited about and interested in, rather than assigning them.	**Revised the description of the practice for readability and understanding.**
Creating a culture to share knowledge altruistically	Creating awareness among contributors and involving altruistic philosophy is the main reason behind initiating an OSS project. Knowledge sharing can be introduced as a culture in OSS to stabilise ecosystem and selfless gesture of doing good to the society. As an example of intrinsic motivation, Google consists of a user community mainly of software engineers. The knowledge is shared by answering questions and helping solve problems that other software engineer post, without being compensated.	**Revised the description of the practice for readability and understanding.**
Making project documentation publicly accessible	Public access provides visibility to ongoing project documentation. This can be related to the communicating changes in the documentation on the fly. But it is more of an adoption of a culture or environment where documentation is given public access to be viewed by the contributors of the project. This practices also enables the building ans strengthening of the ecosystem. Project document should have an open access to all project community. When communication is public and accessible, anybody can read past archives to get up to speed by going though mailing lists, blogs and participate. For example, public communication can be as simple as directing people to open an issue instead of emailing directly to project leader or commenting on project's blog. Further, setting up a mailing list, or creating a Twitter account, Slack, or IRC channel for people to talk about the project..	**Revised the description of the practice for readability and understanding.**
Fostering an open-minded culture towards diverse types of contributions	OSS project governing teams and other contributors should adopt a culture to be acceptable to all kinds of contributions including small ones. This kind of open culture encourages contributors who are just casual contributors but their efforts are recognised and accepted. For example, the types of contributions to accept can be a bug report or a small fix making it easier for casual contributors to contribute.	**Revised the description of the practice for readability and understanding.**

	To encourage mentorship of new comers	Mentoring contributors enthusiastic to work on the project for a head start and submit quality contributions. This practice is more useful to overcome knowledge barriers faced by contributors mainly who are new to the project. Someone is enthusiastic about the project, but needs a bit of polishing can be considered for mentoring through their first contribution. For example, resist fixing easy (non-critical) bugs. Instead, use them as an opportunity to recruit new contributors and mentor someone who would like to contribute.	**Revised the description of the practice for readability and understanding.**
Governance and Leadership (8)	Frequent updation of project documents	This practice is useful for the removal of knowledge barriers. It is the responsibility of the contributor working on the project to ensure that documents are up to date. In case of problem or clarification required on the process it should be brought to the attention of team leaders. This can be related to culture category of practices but having a culture solely would not fulfil the need of contributors when it comes to accessibility and correctness of knowledge available. The knowledge centralisation and updation has to be monitored by team leaders and its accessibility to contributors and teams. Also team leads or people governing OSS project are generally more knowledgeable to update the documents at the central location. For example centralisation is explained by inspecting the governance structures in Ericsson, which is argued to be similar to OSS, and where a centralised approach is implemented to secure quality [128]. The situation of knowledge loss faced was because of the recurrent movement of resources in and out of products and constantly changing business needs. In such a situation, the adoption of a centralised approach helped in architectural knowledge stability and its availability to new developers and teams. Centralisation also involves actively maintaining the project documents.	**Revised the description of the practice for readability and understanding.**
	Enabling a strategy of successor identification	Identification of successors and involving contributors as co-owners with relevant expertise knowledgeable on the work of other contributors is presented as a method to reduce the risk associated with developer turnover [10]. The files with a successor were not at risk of abandonment even when the owning developer left. A successor was there to perform maintenance tasks [10]. For example, adopting real-time visualisation of resources can facilitate in the identification of successors. Project leaders, owners or team leaders in OSS may relocate resources on projects based on the visualisation of resources when there are less resources than required. Decisions to use a technology tool that depicts the real-time visualisation of resources is mainly upon the main leaders of the project	**Revised the description of the practice for readability and understanding.**

		Another example is to develop a Transactive Memory Systems / Knowledge map / Knowledge portals. The team leaders would be able to keep the track and identifying people who are knowledgeable on the work of others. Transactive Memory System development within the teams is based on knowledge location, the usage of the developer mailing list and knowledge credibility. An insight into the area of expertise members, a knowledge map or directory can be used on the project website internal knowledge portal, serving as a way to help knowledge workers familiarize themselves with their colleagues knowledge, K Portals are rapidly evolving into broad-based platforms for supporting a wide range of knowledge worker (KW) tasks	
	Proactive assignment of varying tasks to non-cores	The skill set of core contributors is better than non-cores. Cores make 80% of the knowledge contributions in OSS projects. In order to uniformly distribute knowledge from cores to non-cores, the team leads can assign different tasks to non-cores proactively. This falls under government and leadership because the team leads can control the extent of knowledge that is shared with non-cores. proactive assignment of different tasks to non-cores can be governed by leaders in OSS projects. As indicated that contributors who modify code from other contributors stay longer on the project [28]. This will create a balance of equal development of skills on the OSS project. For example, contributors who normally perform documentation tasks should be assigned some coding tasks.	**Revised the description of the practice for readability and understanding.**
	Advocating diversification of core contributors specialisation	For an OSS project to survive, a diversity of core developers is required [132]. When a key contributor abandoned an OSS project it revealed a very fluctuating proportion of developer contribution. A significant imbalance between the contribution and the response from the developers' community was noticed. The reason for the dying project was that a diversity of core contributors were missing from the project [132]. Diversity of core contributors also relates to the underlying concept of uniform knowledge distribution stated next. Balance between the contribution submitted and response from specialised core developers in community. Form a "core team" of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct). As an example, job rotation allows for knowledge exchange and transfer while people take turn to work in different job roles, tasks, and domains. Job rotation legitimises experience that allows people in the organisation to work in diverse knowledge domains. Some development practices, such as pair programming, facilitate knowledge sharing between peers, while job rotation helps knowledge spread throughout the project or organisation.	**Revised the description of the practice for readability and understanding.**

Appendix C

The Survey Instrument and Data Collection

C.1 Survey Instrument

Open Source Software Community Survey

How would you like to learn about emerging best practices for Open Source Software communities while also being part of the expert group that helps to shape those practices? If this is of interest to you then please rate the knowledge exchange practices contained in the following survey. It is not a lengthy survey and you will be finished in a few short minutes.

Your participation is voluntary and confidential. All responses are entirely anonymous. This work is part of a University research programme being undertaken at Dublin City University, Ireland.

This objective of this work is to strengthen Open Source Software communities and I hope that you will contribute to this effort.

Further information on this research project is available at goo.gl/j8ifQR and if you wish to reach out to discuss my work then please e-mail: mehvish.rashid2@mail.dcu.ie

Kind regards,

Mehvish Rashid
Dublin City University
Ireland

NEXT

Page 1 of 8

Never submit passwords through Google Forms.

Open Source Software Community Survey

*Required

Consent

Please tick this box to indicate you consent to your input to this survey being used anonymously and you understand that your involvement is voluntary. *

☐ I consent

BACK NEXT

Page 2 of 8

Never submit passwords through Google Forms.

Open Source Software Community Survey

*Required

Please choose as appropriate.

1. In total how many OSS projects have you worked on? *

- ○ 1-5
- ○ 5 -10
- ○ 10+

2. Which of the following activities apply to your OSS work? *

- ☐ Bug reporter
- ☐ Code contributor
- ☐ Maintainer
- ☐ Reviewer
- ☐ Committer
- ☐ Document writer/ editor
- ☐ Tester
- ☐ Integrator
- ☐ Other: _____

3. How many years have you been contributing to OSS projects? *

- ◯ 1 to 5 Years
- ◯ 5 to 10 Years
- ◯ 10+ Years

4. How much computer programming experience do you have? *

- ◯ Less than 1 Year
- ◯ 1 to 2 Years
- ◯ 2 to 5 Years
- ◯ 5 to 10 Years
- ◯ 10+ Years

BACK NEXT Page 3 of 8

Open Source Software Community Survey

*Required

Provide your rating of the effectiveness of the following knowledge exchange practices on OSS projects.

1. Encourage pair programming and shared code ownership *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

2. Promote a policy that encourages peer review contributions from all project roles (irrespective of their role or seniority) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

3. Use bug labeling so that contributors can effectively select tasks and make contributions (e.g. "suited for newcomers", "Feature xyz") *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

4. Ensure the presence of testing artefacts (e.g. unit tests, test scripts, test cases). *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

5. Label substantial new items of work as "work in progress" indicating an opportunity for more contributions *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

6. Introduce non-restrictive commit access to non-core contributors (where this is appropriate) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

7. Establish explicit code review guidelines (e.g. a maximum time limit for code review feedback for all contributors) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

8. Publish findings of post-mortem reviews (lesson learned on the project) to all contributors *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

BACK NEXT

Page 4 of 8

Open Source Software Community Survey

*Required

Provide your rating of the effectiveness of the following knowledge exchange practices on OSS projects.

9. Build a supportive community for conflict resolution *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

10. Explicitly identify contributors to reach out for queries, solutions, and support. *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

11. Allow self-organization of project roles *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

12. Promote an altruistic knowledge sharing culture *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

13. Make project documentation publicly accessible (e.g. public access to archives, mailing list, and to open an issue) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

14. Foster an open-minded culture towards diverse types of contributions *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

15. Encourage mentorship of newcomers *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

BACK NEXT

Page 5 of 8

Open Source Software Community Survey

*Required

Provide your rating of the effectiveness of the following knowledge exchange practices on OSS projects.

16. Establish core to non-core knowledge sharing practices (e.g. interviews). *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Higly effective

17. Have an explicit communication protocol between contributors and integrators (responsible for merging code to the main branch). *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

18. Leaders managing the project respond within a set time limit to queries, reported issues, and pull requests. *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

19. Encourage open discussion to resolve matters concerning the project community. *

0	1	2	3	4	5	6	7	8	9	10

20. Promote the documentation of solutions to commonly experienced problems *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

BACK NEXT Page 6 of 8

Open Source Software Community Survey

*Required

Provide your rating of the effectiveness of the following knowledge exchange practices on OSS projects.

21. Update project documents frequently (e.g. document changes on process, architecture, domain) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

22. Identify successors for key code contributors *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

23. Proactive assignment of varying tasks to non-core contributors *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Higly effective

24. Advocate diversification of core contributor specialisation *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

25. Distribute project leadership to the wider support community when primary leaders are absent. *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

26. Document role progression process (e.g. from contributor to integrator to maintainers). *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

27. Document project rules and policies (e.g. the type of contributions required, policy to review contributions) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

28. Establish training mechanisms (e.g. on-the-job training and shadowing) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

BACK NEXT

Page 7 of 8

Open Source Software Community Survey

*Required

Provide your rating of the effectiveness of the following knowledge exchange practices on OSS projects.

29. Create a knowledge contribution recognition program (e.g. a reward structure) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

30. Reward active knowledge contributors with project seniority *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

31. Promote a culture of appreciation for knowledge contributors (e.g. words of appreciation through blogs and newsletters) *

	0	1	2	3	4	5	6	7	8	9	10	
Not effective	○	○	○	○	○	○	○	○	○	○	○	Highly effective

32. Please state any other knowledge relevant practices that are not listed above and in your opinion can benefit the OSS project(s)?

Your answer

BACK SUBMIT

Page 8 of 8

Open Source Software Community Survey

Thank you for your time and valuable feedback!

We plan to include the results of this survey in a scientific publication, if you would like to receive updates, please send an e-mail to: mehvish.rashid2@mail.dcu.ie

C.2 Data Collected Through Survey Instrument

(Consent, Profile Questions 1-4, and Practice Questions 1-4)

Profile Questions 1-4					Questions on PKR Practices 1-4			
Please tick this box to indicate you consent to your input to this survey being used anonymously and you understand that your involvement is voluntary.	1. In total how many OSS projects have you worked on?	2. Which of the following activities apply to your OSS work?	3. How many years have you been contributing to OSS projects?	4. How much computer programming experience do you have?	1. Encourage pair programming and shared code ownership	2. Promote a policy that encourages peer review contributions from all project roles (irrespective of their role or seniority)	3. Use bug labeling so that contributors can effectively select tasks and make contributions (e.g. "suited for newcomers", "Feature xyz")	4. Ensure the presence of testing artefacts (e.g. unit tests, test scripts, test cases).
I consent	10+	Bug reporte	10+ Years	10+ Years	8	7	10	8
I consent	10+	Bug reporte	10+ Years	10+ Years	6	10	9	10
I consent	10+	Bug reporte	10+ Years	10+ Years	3	9	7	9
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	5	7	7	9
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	10	10	10	10
I consent	5 -10	Bug reporte	1 to 5 Years	5 to 10 Year	8	7	8	8
I consent	1-5	Bug reporte	10+ Years	10+ Years	7	10	10	9
I consent	10+	Bugreporte	5 to 10	5 to 10	7	7	10	10
I consent	10+	Bug reporte	10+ Years	10+ Years	2	6	2	6
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	7	7	3	2
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	10	10	9	8
I consent	10+	Bug reporte	10+ Years	10+ Years	2	4	8	10
I consent	10+	Bug reporte	10+ Years	10+ Years	6	9	8	10
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	10	8	7	10
I consent	10+	Bug reporte	10+ Years	10+ Years	10	10	8	10
I consent	10+	Bug reporte	10+ Years	10+ Years	7	7	8	8
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	7	10	8	10
I consent	10+	Bug reporte	1 to 5 Years	2 to 5 Years	8	9	7	7
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	2	9	5	9
I consent	10+	Bug reporte	1 to 5 Years	10+ Years	7	5	7	10
I consent	1-5	Bug reporte	5 to 10 Yea	10+ Years	5	5	8	8
I consent	5 -10	Bug reporte	10+ Years	10+ Years	7	6	8	9
I consent	10+	Bug reporte	1 to 5 Years	10+ Years	5	8	10	9
I consent	10+	Bug reporte	1 to 5 Years	2 to 5 Years	5	5	10	8
I consent	5 -10	Bug reporte	1 to 5 Years	5 to 10 Year	9	8	7	8
I consent	10+	Bug reporte	1 to 5 Years	2 to 5 Years	10	10	8	8
I consent	1-5	Code contri	10+ Years	10+ Years	0	0	10	10
I consent	1-5	Code contri	1 to 5 Years	5 to 10 Year	8	9	9	7
I consent	1-5	Code contri	1 to 5 Years	5 to 10 Year	8	5	10	8
I consent	1-5	Code contri	1 to 5 Years	2 to 5 Years	7	9	9	7
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	10	10	10	10
I consent	1-5	Bug reporte	1 to 5 Years	5 to 10 Year	6	9	6	8
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	7	10	7	10

I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	7	8	4	10
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	1	7	5	8
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	5	8	7	9
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	6	8	10	9
I consent	10+	Bug reporte	1 to 5 Years	2 to 5 Years	0	10	10	8
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	8	3	5	5
I consent	10+	Bug reporte	1 to 5 Years	2 to 5 Years	2	10	8	7
I consent	1-5	Bug reporte	5 to 10 Yea	10+ Years	5	10	9	8
I consent	5 -10	Bug reporte	5 to 10 Yea	2 to 5 Years	6	8	10	10
I consent	1-5	Code contri	1 to 5 Years	5 to 10 Year	1	8	10	7
I consent	10+	Bug reporte	10+ Years	10+ Years	2	6	4	9
I consent	1-5	Reviewer, C	1 to 5 Years	2 to 5 Years	10	10	10	10
I consent	5 -10	Code contri	10+ Years	10+ Years	2	8	1	10
I consent	1-5	Bug reporte	1 to 5 Years	2 to 5 Years	8	8	8	8
I consent	10+	Bug reporte	10+ Years	10+ Years	9	2	4	8
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	5	6	6	10
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	10	10	10	10
I consent	10+	Bug reporte	10+ Years	5 to 10 Year	5	8	10	10
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	4	5	7	9
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	8	8	9	10
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	8	9	8	7
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	8	7	10	6
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	4	3	6	7
I consent	10+	Bug reporte	10+ Years	10+ Years	5	8	8	10
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	5	8	10	10
I consent	1-5	Bug reporte	1 to 5 Years	2 to 5 Years	7	9	8	8
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	7	8	10	10
I consent	10+	Bug reporte	10+ Years	10+ Years	8	4	6	7
I consent	1-5	Bug reporte	1 to 5 Years	10+ Years	1	5	9	9
I consent	5 -10	Bug reporte	1 to 5 Years	5 to 10 Year	10	8	10	10
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	0	10	10	10
I consent	1-5	Bug reporte	1 to 5 Years	2 to 5 Years	9	8	8	8
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	8	9	8	10
I consent	5 -10	Code contri	5 to 10 Yea	10+ Years	5	9	7	10
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	8	10	9	10
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	8	6	10	6
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	10	8	7	8
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	8	10	8	10
I consent	1-5	Bug reporte	1 to 5 Years	5 to 10 Year	8	5	8	8
I consent	5 -10	Bug reporte	10+ Years	10+ Years	10	7	5	7
I consent	10+	Bug reporte	5 to 10 Yea	5 to 10 Year	7	9	3	10
I consent	10+	Bug reporte	10+ Years	10+ Years	2	3	8	10
I consent	10+	Bug reporte	10+ Years	10+ Years	7	9	7	10
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	0	0	7	6
I consent	1-5	Code contri	1 to 5 Years	10+ Years	8	8	7	9
I consent	5 -10	Bug reporte	5 to 10 Yea	5 to 10 Year	8	6	3	5
I consent	5 -10	Bug reporte	5 to 10 Yea	10+ Years	8	8	6	7
I consent	1-5	Bug reporte	1 to 5 Years	5 to 10 Year	8	8	8	8
I consent	10+	Bug reporte	5 to 10 Yea	10+ Years	5	4	8	10
I consent	10+	Bug reporte	10+ Years	10+ Years	10	10	10	10

I consent	10+	Bug reporte	10+ Years	10+ Years	6	7	2	4
I consent	1-5	Bug reporte	10+ Years	10+ Years	8	8	5	10
I consent	1-5	Bug reporte	1 to 5 Years	10+ Years	6	9	7	7
I consent	1-5	Bug reporte	1 to 5 Years	10+ Years	10	10	10	10
I consent	10+	Bug reporte	1 to 5 Years	2 to 5 Years	10	10	8	10
I consent	10+	Bug reporte	1 to 5 Years	2 to 5 Years	8	8	10	7
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	5	8	7	8
I consent	5 -10	Code contri	5 to 10 Yea	10+ Years	5	8	10	10
I consent	10+	Bug reporte	10+ Years	10+ Years	0	8	3	10
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	10	8	8	10
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	7	9	7	4
I consent	1-5	Bug reporte	1 to 5 Years	5 to 10 Year	5	6	9	10
I consent	10+	Bug reporte	10+ Years	10+ Years	7	7	10	7
I consent	10+	Bug reporte	10+ Years	10+ Years	5	10	10	10
I consent	10+	Bug reporte	10+ Years	10+ Years	8	5	7	2
I consent	10+	Bug reporte	10+ Years	10+ Years	5	10	5	10
I consent	5 -10	Bug reporte	5 to 10 Yea	10+ Years	8	5	8	7
I consent	1-5	Bug reporte	1 to 5 Years	2 to 5 Years	5	7	8	8
I consent	5 -10	Bug reporte	5 to 10 Yea	5 to 10 Year	8	8	10	10
I consent	5 -10	Bug reporte	10+ Years	10+ Years	7	10	9	4
I consent	10+	Bug reporte	10+ Years	10+ Years	10	10	10	7
I consent	1-5	Code contri	1 to 5 Years	10+ Years	10	10	10	10
I consent	1-5	Code contri	1 to 5 Years	10+ Years	6	8	6	7
I consent	10+	Bug reporte	1 to 5 Years	10+ Years	5	8	10	10
I consent	1-5	Code contri	1 to 5 Years	10+ Years	8	2	8	10
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	10	8	10	7
I consent	5 -10	Code contri	5 to 10 Yea	5 to 10 Year	8	6	9	10
I consent	1-5	Code contri	10+ Years	10+ Years	6	7	8	6
I consent	10+	Bug reporte	10+ Years	10+ Years	3	10	8	10
I consent	1-5	Bug reporte	10+ Years	10+ Years	3	7	7	8
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	5	4	9	8
I consent	10+	Bug reporte	10+ Years	5 to 10 Year	5	5	6	7
I consent	10+	Bug reporte	10+ Years	10+ Years	5	8	8	10
I consent	1-5	Bug reporte	5 to 10 Yea	10+ Years	5	6	10	10
I consent	1-5	Code contri	1 to 5 Years	1 to 2 Years	3	6	6	6
I consent	1-5	Bug reporte	1 to 5 Years	10+ Years	6	7	8	9
I consent	1-5	Bug reporte	10+ Years	10+ Years	6	2	3	8
I consent	1-5	Bug reporte	5 to 10 Yea	10+ Years	5	8	10	9
I consent	10+	Bug reporte	10+ Years	10+ Years	9	10	10	10
I consent	10+	Bug reporte	1 to 5 Years	5 to 10 Year	5	10	10	10
I consent	1-5	Code contri	1 to 5 Years	5 to 10 Year	8	8	8	8
I consent	5 -10	Bug reporte	1 to 5 Years	2 to 5 Years	9	9	7	8
I consent	1-5	Code contri	1 to 5 Years	1 to 2 Years	5	9	9	4

C.3 Data Collected Through Survey Instrument (Questions 5-13)

Questions on PKR Practices 5-13								
5. Label substantial new items of work as "work in progress" indicating an opportunity for more contributions	**6. Introduce non-restrictive commit access to non-core contributors (where this is appropriate)**	**7. Establish explicit code review guidelines (e.g. a maximum time limit for code review feedback for all contributors)**	**8. Publish findings of post-mortem reviews (lesson learned on the project) to all contributors**	**9. Build a supportive community for conflict resolution**	**10. Explicitly identify contributors to reach out for queries, solutions, and support.**	**11. Allow self-organization of project roles**	**12. Promote an altruistic knowledge sharing culture**	**13. Make project documentation publicly accessible (e.g. public access to archives, mailing list, and to open an issue)**
3	8	4	6	7	7	7	6	10
5	10	8	6	10	8	9	10	10
5	10	4	2	9	6	8	10	10
5	2	5	8	6	8	7	10	10
7	4	7	8	8	7	5	7	10
9	10	8	9	9	10	9	8	9
10	1	8	7	10	9	9	9	9
9	5	5	4	10	8	9	9	10
1	0	3	6	6	3	5	7	3
7	2	2	2	6	5	4	6	4
7	6	5	6	8	7	8	9	8
5	8	5	8	5	8	8	8	8
4	2	7	7	10	8	8	9	10
7	6	6	10	8	9	7	8	9
8	7	8	7	5	4	8	10	10
7	6	5	5	10	7	5	10	10
4	1	4	5	3	2	2	6	8
6	4	7	5	9	10	8	9	9
5	4	8	9	4	7	7	7	10
6	8	6	7	7	8	8	10	10
5	8	8	8	9	5	5	7	9
6	5	8	7	9	8	8	9	9
5	2	9	8	7	6	7	5	7
9	8	7	9	7	8	8	8	10
5	7	3	7	6	4	7	8	10
6	9	5	3	4	6	4	10	10
10	5	5	5	5	10	10	5	5
9	9	8	9	9	9	6	10	10
2	6	2	3	0	5	10	4	10
7	7	7	5	9	8	8	9	9
10	6	10	6	10	6	8	10	10
9	8	9	9	9	8	8	9	9
10	3	10	7	7	0	7	10	10
8	7	9	9	9	9	9	10	10
4	6	6	9	8	6	6	6	10

5	5	7	8	6	8	8	5	9
8	5	6	10	7	5	9	7	10
10	6	2	2	10	8	10	10	10
7	7	7	7	8	9	4	10	9
5	5	10	9	3	10	5	8	10
8	2	10	2	10	3	8	8	10
10	10	10	6	10	6	7	10	7
3	0	6	8	8	10	10	7	9
3	4	3	4	7	7	6	7	10
10	10	7	8	10	7	8	8	10
1	3	1	1	10	9	10	10	10
8	5	8	8	8	5	5	6	8
5	1	2	9	5	9	8	9	10
6	10	5	5	8	8	8	10	10
10	9	10	10	10	10	10	10	10
8	0	5	8	10	10	6	10	10
3	2	2	3	7	8	7	10	10
6	4	8	5	9	8	9	10	10
8	7	9	6	8	7	7	8	9
10	9	10	7	8	9	7	7	10
4	5	3	4	5	7	6	7	9
5	2	7	8	10	10	5	10	7
8	0	5	8	10	10	5	10	10
8	5	8	9	8	4	5	8	8
3	8	4	5	8	10	6	8	10
5	1	6	10	9	9	5	4	10
6	1	6	3	3	8	8	7	9
10	7	8	10	10	10	7	8	10
7	4	3	6	10	10	6	10	10
7	6	8	7	8	7	6	9	7
7	7	8	10	9	9	9	10	10
5	8	4	5	5	7	5	10	10
4	2	4	7	10	9	7	9	10
3	7	4	7	8	4	7	10	10
3	6	8	7	8	9	9	8	9
2	7	9	8	7	7	2	9	10
8	8	8	8	10	10	10	10	10
9	7	5	10	5	5	8	10	10
5	0	5	8	6	6	5	8	9
3	2	3	8	10	10	8	10	10
7	6	5	9	10	9	7	7	10
6	0	0	0	6	5	0	8	7
7	7	7	7	6	5	7	8	9
3	4	3	8	7	8	6	7	9
7	8	6	6	7	7	7	8	9
8	8	8	8	8	8	8	8	8
6	0	4	7	7	7	7	9	10
5	10	10	10	10	10	10	10	10
2	5	0	3	2	3	0	8	9
5	3	10	10	9	8	10	5	9

9	3	7	8	7	5	7	8	8
10	10	10	10	7	10	7	10	10
10	7	0	10	10	10	5	10	10
10	10	5	10	7	10	10	10	10
4	3	7	9	8	10	6	9	10
7	2	5	5	8	5	7	5	10
2	0	0	5	2	5	8	8	10
6	0	10	10	10	10	9	10	10
7	6	3	5	9	9	9	9	9
1	3	3	6	6	9	7	10	10
2	0	5	0	0	7	10	7	10
10	5	10	5	5	10	8	10	10
6	2	7	8	9	10	7	8	10
5	5	5	5	7	5	8	8	10
5	1	9	5	10	8	1	5	10
8	5	6	7	5	3	5	9	9
7	7	6	9	8	6	5	8	10
8	9	7	9	9	7	5	7	10
6	7	8	7	8	10	6	10	10
10	0	10	10	10	10	0	10	10
7	6	8	8	7	7	6	9	8
8	8	2	7	7	7	5	8	8
10	0	8	8	7	5	8	8	8
7	7	7	6	10	6	10	10	8
1	2	9	8	8	6	7	8	9
8	9	8	7	9	8	9	9	9
10	7	0	10	7	5	5	10	10
3	3	3	3	7	7	7	8	9
8	3	2	2	6	8	9	6	9
4	9	7	9	7	9	6	10	10
3	5	8	10	10	10	8	10	10
5	8	9	8	8	5	7	7	10
10	1	4	7	7	7	8	9	10
2	2	3	4	8	7	7	10	9
7	5	3	6	8	9	9	9	10
10	3	8	9	9	10	6	9	10
6	6	9	9	10	10	10	10	10
10	5	10	5	10	10	10	10	10
5	2	7	9	8	2	2	10	9
6	3	6	6	6	7	6	6	9
6	6	9	8	8	8	3	6	6

C.4 Data Collected Through Survey Instrument Questions 13-22

Questions on PKR Practices 14-22								
14. Foster an open-minded culture towards diverse types of contributions	15. Encourage mentorship of newcomers	16. Establish core to non-core knowledge sharing practices (e.g. interviews).	17. Have an explicit communication protocol between contributors and integrators (responsible for merging code to the main branch).	18. Leaders managing the project respond within a set time limit to queries, reported issues, and pull requests.	19. Encourage open discussion to resolve matters concerning the project community.	20. Promote the documentation of solutions to commonly experienced problems	21. Update project documents frequently (e.g. document changes on process, architecture, domain)	22. Identify successors for key code contributors
7	7	7	6	6	8	10	8	3
10	10	3	6	3	8	10	8	2
10	10	6	3	0	8	10	3	7
7	9	8	9	6	7	9	9	5
10	7	5	6	5	8	8	10	6
8	9	6	8	10	8	7	9	7
7	9	7	7	7	10	9	9	8
10	10	3	3	7	7	9	7	8
6	3	7	4	1	2	4	4	3
8	9	2	4	2	4	4	8	6
9	10	6	8	9	6	9	8	7
8	10	5	2	9	8	10	10	8
8	8	8	8	3	8	7	6	6
10	10	7	7	8	10	10	10	7
6	7	2	4	6	8	10	8	2
10	10	7	8	5	10	10	10	7
7	8	3	0	10	3	7	5	6
9	8	6	7	7	8	9	9	8
6	8	4	5	6	6	8	7	8
7	7	8	6	8	6	7	6	8
10	6	5	6	5	7	6	7	6
8	8	8	9	7	9	8	6	6
3	2	2	8	9	7	9	8	10
7	9	8	8	8	10	9	10	8
10	9	8	5	8	7	7	7	6
7	10	3	0	6	10	10	7	4
5	10	5	5	0	5	10	10	10
10	10	9	9	9	8	10	10	10
0	2	2	3	0	8	6	8	8
9	9	7	8	7	7	8	8	7
10	10	10	10	10	10	10	6	10
8	8	9	7	10	7	9	9	8
10	7	0	0	7	3	10	7	0

10	7	7	6	3	7	8	8	7
4	8	4	4	4	4	4	0	0
8	7	8	8	9	9	9	9	8
5	7	4	1	4	6	9	9	8
10	10	0	10	7	10	10	10	5
9	2	5	5	8	8	3	4	8
1	10	5	9	10	0	9	5	5
10	8	2	4	6	10	10	10	9
7	8	7	3	7	7	6	6	4
9	6	3	7	5	9	10	10	8
8	6	3	5	5	8	8	10	2
10	10	5	5	5	7	8	6	7
7	6	1	1	2	5	8	6	3
9	8	7	8	8	8	8	8	6
1	7	4	6	3	7	7	3	2
8	5	5	5	5	8	9	10	5
10	10	10	10	10	10	10	10	10
10	10	5	8	8	10	10	8	8
7	9	3	4	8	7	10	7	8
10	10	7	7	8	8	9	10	8
8	7	5	7	6	7	7	8	6
9	5	3	7	6	6	5	7	8
4	3	2	4	5	6	7	6	4
7	7	5	6	8	6	10	10	8
10	10	2	5	2	8	10	8	8
9	8	8	8	8	9	9	9	8
10	8	6	7	5	7	10	8	9
8	9	3	6	2	6	9	6	9
7	6	2	4	4	7	9	7	8
10	10	7	7	8	9	9	9	9
10	10	0	3	10	10	10	10	6
9	9	8	8	9	9	9	8	7
10	10	7	7	8	10	8	9	8
6	6	4	4	7	8	10	5	7
10	8	5	7	8	9	10	7	5
10	10	6	7	3	9	10	9	8
10	7	8	8	8	9	9	6	7
10	10	7	9	10	10	10	8	7
10	10	5	7	8	8	10	10	10
5	7	3	5	8	2	10	10	5
9	5	5	5	6	8	10	9	5
10	10	2	8	3	8	8	8	5
9	10	7	7	5	9	9	9	6
6	6	0	7	0	7	9	6	7
9	7	9	5	6	6	6	5	5
9	9	10	6	7	7	8	7	3
7	8	6	7	7	8	7	7	5
8	8	8	8	8	8	8	8	8
9	5	10	4	9	8	10	10	4
10	10	10	10	10	10	10	10	10

0	10	5	3	7	2	7	10	7
9	7	5	8	8	5	9	8	8
6	8	7	6	7	7	7	4	6
10	10	10	10	10	10	10	10	10
10	10	10	10	6	10	10	4	10
7	7	7	10	5	7	10	10	7
9	10	9	9	9	9	10	10	8
9	9	2	6	2	8	8	9	5
3	10	4	5	4	5	10	10	7
10	8	7	10	10	10	10	10	10
9	8	7	5	3	7	10	9	7
10	8	6	9	2	6	2	6	7
5	7	5	5	7	10	10	0	0
10	10	10	10	10	10	10	10	8
7	7	3	5	8	9	9	8	9
8	8	5	10	5	8	10	10	6
10	10	4	8	8	8	10	7	7
8	8	5	7	6	4	7	8	9
9	10	7	9	7	9	10	10	8
10	8	6	6	3	6	8	7	8
10	10	8	8	10	9	7	8	9
10	10	10	10	8	5	10	10	10
8	7	7	7	6	7	9	8	7
5	7	8	9	2	8	7	7	3
8	4	10	8	8	10	8	8	10
6	9	4	6	3	6	4	7	7
9	9	6	9	8	8	9	8	7
9	9	8	7	8	9	8	10	8
8	10	3	10	0	8	10	8	10
7	6	5	5	1	8	8	7	7
5	7	5	6	3	6	9	8	7
4	6	6	5	6	5	9	7	4
10	8	4	5	7	10	10	6	0
4	6	5	5	6	7	9	9	8
9	6	5	4	7	8	9	8	5
9	7	6	5	5	7	8	7	5
8	8	4	3	4	7	5	5	8
9	8	8	10	6	8	10	9	10
10	10	9	10	9	10	9	9	9
10	10	9	8	9	10	10	9	10
10	10	7	10	8	9	9	10	5
4	4	5	6	7	7	8	8	5
5	4	3	4	7	8	7	6	7

C.5 Data Collected Through Survey Instrument (Question 23-32)

Questions on PKR Practices 23-32									
23. Proactive assignment of varying tasks to non-core contributors	**24. Advocate diversification of core contributor specialisation**	**25. Distribute project leadership to the wider support community when primary leaders are absent.**	**26. Document role progression process (e.g. from contributor to integrator to maintainers).**	**27. Document project rules and policies (e.g. the type of contributions required, policy to review contributions)**	**28. Establish training mechanisms (e.g. on-the-job training and shadowing)**	**29. Create a knowledge contribution recognition program (e.g. a reward structure)**	**30. Reward active knowledge contributors with project seniority**	**31. Promote a culture of appreciation for knowledge contributors (e.g. words of appreciation through blogs and newsletters)**	**32. Please state any other knowledge relevant practices that are not listed above and in your opinion can benefit the OSS project(s)?**
1	2	7	4	4	6	3	4	6	Corporate sponsorships
2	7	7	6	7	1	2	2	6	
2	8	9	2	6	1	0	0	9	?
2	4	2	3	8	2	1	2	6	N/A
8	5	5	5	8	4	0	5	3	""
9	7	9	5	7	6	8	10	7	-
6	6	8	9	6	6	8	8	8	-
4	3	7	1	2	2	7	6	9	
0	4	4	3	5	1	0	4	3	...
2	7	3	1	3	4	5	2	7	N/A
10	7	8	6	8	9	7	9	10	Have a CoC
5	5	8	5	8	8	3	8	8	N/A
3	4	3	3	9	3	2	5	7	No opinion.
7	9	6	5	7	3	7	9	10	Make the project more known
4	0	0	0	2	8	10	3	10	Collaborate with academic
5	7	10	5	7	5	2	5	10	n/a
7	6	7	8	4	0	7	5	6	-
6	6	4	6	9	5	8	9	8	Publish roadmaps and plans
7	6	4	7	8	5	4	9	9	N/A
7	8	7	4	7	3	2	4	5	Staying active until others are
7	6	8	6	7	7	6	7	6	Improvement of OSS usability
7	7	7	7	5	8	2	8	5	none
3	5	8	7	4	3	7	7	7	Automated Release generation,
8	6	8	7	7	6	7	6	7	.
7	7	9	5	5	7	6	7	7	Be kind
0	5	8	0	0	2	4	4	1	Usage statistics can help shift
5	0	5	5	10	5	10	10	10	Value autonomy. It leads to
8	8	8	9	8	9	9	9	9	These are all
8	6	2	4	2	8	8	10	9	Keeping projects free of
5	7	8	10	8	2	2	8	6	

6	10	10	10	10	6	8	10	10	N/A
9	7	9	9	7	8	9	8	8	A pattern more common
0	5	7	8	10	5	6	0	7	
3	3	4	3	3	3	6	5	6	
0	0	0	0	0	0	0	0	0	
6	7	8	9	8	7	7	6	7	
6	7	6	0	6	5	7	10	10	
10	10	10	2	10	0	10	8	10	
6	5	7	2	3	0	6	6	8	Identify your contributors and
5	2	2	0	6	7	0	5	2	
5	5	1	10	10	2	5	5	5	
6	5	6	6	10	8	8	7	6	
4	10	9	0	5	1	0	9	9	
2	5	2	2	7	2	2	4	9	
6	8	9	7	7	8	10	10	10	
5	5	5	0	8	0	3	7	4	
6	6	5	6	8	6	8	6	8	
3	0	6	0	0	6	0	5	8	
5	5	8	5	5	5	5	5	5	
5	10	10	10	10	10	10	10	10	
8	8	6	8	10	6	5	8	10	
6	5	7	3	3	2	2	2	3	
8	9	6	7	7	6	8	8	10	
7	7	6	7	7	4	8	7	6	
8	5	7	7	8	5	5	5	5	
3	5	4	6	5	3	3	5	6	
9	8	8	10	8	6	3	6	5	
2	5	2	2	8	5	2	5	8	
8	8	9	8	9	9	10	9	9	
8	10	9	10	8	4	7	10	10	
1	5	3	6	8	6	5	7	6	
3	4	7	1	7	2	1	3	7	
6	9	9	5	7	6	6	6	7	
4	7	6	5	10	0	0	8	10	
7	8	7	7	8	8	8	8	8	
8	8	8	8	9	10	10	8	9	
6	5	7	7	10	5	4	8	6	
6	4	5	5	4	3	4	5	7	
6	7	7	8	10	7	7	8	8	Code of conduct enforcement
7	7	8	6	7	7	9	9	8	
5	10	9	5	8	5	7	7	9	
0	5	5	2	3	10	10	10	10	
5	7	8	3	2	4	5	5	3	
4	4	4	4	6	5	6	7	7	
7	6	8	5	9	3	4	7	9	
6	8	8	9	9	6	10	9	8	having code owned neutrally in

0	0	3	0	7	0	7	5	8	Related to some of the ones
6	6	7	7	5	5	5	5	5	
2	6	5	7	6	8	6	8	9	
7	5	6	4	4	5	4	8	7	
8	8	8	8	8	8	8	8	8	
8	6	6	7	5	5	3	6	9	
10	10	10	10	10	10	10	10	10	Strong community
7	8	4	2	4	6	7	2	5	
8	8	5	5	3	3	5	5	5	Actually, I took some ideas to
4	6	4	7	5	8	7	7	7	
10	10	10	10	10	10	10	10	10	
10	10	10	0	8	10	10	10	10	
7	7	5	5	10	8	10	10	10	
7	8	9	9	8	9	9	10	10	
6	5	8	5	7	2	5	6	5	
6	3	0	4	4	4	4	8	4	authority should be based on
10	9	10	8	10	7	8	8	10	Enable the core team to discuss
8	6	8	7	6	8	3	2	6	
6	3	7	9	9	7	6	7	9	Adding people to public
0	0	7	0	7	0	0	0	5	
5	2	0	5	10	10	5	5	10	Documentation should be
7	7	5	6	7	6	8	9	10	
5	5	5	7	8	5	5	5	7	
8	8	8	6	8	5	8	8	7	
6	5	6	4	6	8	6	7	8	
8	7	8	7	9	6	7	9	8	
8	7	8	4	8	5	0	3	8	
7	8	10	7	8	7	10	7	9	
10	10	10	10	5	5	10	10	10	
7	7	7	6	7	7	6	6	7	
4	5	4	5	7	7	6	6	7	
5	7	8	4	10	7	7	8	8	Rather exhaustive list, but if I
7	6	6	7	6	5	6	6	6	
9	8	4	8	8	7	8	10	8	
8	9	9	8	8	8	7	5	8	
5	8	8	10	10	5	0	7	10	
6	7	5	4	6	6	2	6	7	
5	6	5	7	7	6	5	8	8	Reduce unnecessary
5	5	5	4	6	5	5	6	10	
6	3	3	6	7	6	8	10	8	
5	5	6	6	6	5	7	6	7	
2	6	2	5	6	8	7	7	7	
8	8	2	2	5	6	8	7	8	Github Wiki
7	2	2	3	4	9	10	5	8	Delivering criticism via
9	10	8	10	10	8	5	7	8	
8	9	8	7	9	10	9	9	9	

10	5	5	5	10	5	10	10	10	
2	8	5	8	9	8	8	10	10	
5	6	6	6	4	4	3	2	7	
4	6	7	3	6	3	4	6	7	

Appendix D

Data Analysis

D.1 Description of Knowledge Retention Practices

Table D.1: Practice description against Practice No.

Practice No.	Description
P1	Encourage pair programming and shared code ownership
P2	Promote a policy that encourages peer review contributions from all project roles (irrespective of their role or seniority)
P3	Use bug labelling so that contributors can effectively select tasks and make contributions (e.g. "suited for newcomers", "Feature xyz")
P4	Ensure the presence of testing artefacts (e.g. unit tests, test scripts, test cases).
P5	Label substantial new items of work as "work in progress" indicating an opportunity for more contributions
P6	Introduce non-restrictive commit access to non-core contributors (where this is appropriate)
P7	Establish explicit code review guidelines (e.g. a maximum time limit for code review feedback for all contributors)
P8	Publish findings of post-mortem reviews (lesson learned on the project) to all contributors
P9	Build a supportive community for conflict resolution
P10	Explicitly identify contributors to reach out for queries, solutions, and support.
P11	Allow self-organization of project roles
P12	Promote an altruistic knowledge sharing culture
P13	Make project documentation publicly accessible (e.g. public access to archives, mailing list, and to open an issue)
P14	Foster an open-minded culture towards diverse types of contributions
P15	Encourage mentorship of newcomers
P16	Establish core to non-core knowledge sharing practices (e.g., interviews).
P17	Have an explicit communication protocol between contributors and integrators (responsible for merging code to the main branch).
P18	Leaders managing the project respond within a set time limit to queries, reported issues, and pull requests.
P19	Open discussion to resolve matters concerning the project community.
P20	Promote the documentation of solutions to commonly experienced problems
P21	Update project documents frequently (e.g. document changes on process, architecture, domain)
P22	Identify successors for key code contributors
P23	Proactive assignment of varying tasks to non-core contributors
P24	Advocate diversification of core contributor specialisation
P25	Distribute project leadership to the wider support community when primary leaders are absent.
P26	Document role progression process (e.g. from contributor to integrator to maintainers).
P27	Document project rules and policies (e.g. the type of contributions required, policy to review contributions)
P28	Establish training mechanisms (e.g. on-the-job training and shadowing)
P29	Create a knowledge contribution recognition program (e.g. a reward structure)
P30	Reward active knowledge contributors with project seniority
P31	Promote a culture of appreciation for knowledge contributors (e.g. words of appreciation through blogs and newsletters)