ISSUE TRACKING METRICS AND ASSIGNEE RECOMMENDATION IN SCIENTIFIC SOFTWARE PROJECTS

HODA NAGUIB

Chair of Applied Software Engineering
Institut für Informatik
Technische Universität München

September 2014

Institut für Informatik

Lehrstuhl Für Angewandte Softwaretechnik - Chair for Applied Software Engineering

Issue Tracking Metrics and Assignee Recommendation in Scientific Software Projects

Hoda Mohamed Naguib Gunidi Khalafalla

Vollständiger Abdruck der von der Fakultät für Informatik
der Technischen Universität München zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigten Dissertation.

Vorsitzende(r): Univ.-Prof. Dr. Michael Bader
Prüfer der Dissertation:
1. Univ.-Prof. Bernd Brügge, Ph.D.
2. Univ.-Prof. Dr. Hans Michael Gerndt

Die Dissertation wurde am 23.09.2014 bei der Technischen Universität München
eingereicht und durch die Fakultät für Fakultät für Informatik
am 08.01.2015 angenommen.

Hoda Naguib: *Issue Tracking Metrics and Assignee Recommendation in Scientific Software Projects*, Doktors der Naturwissenschaften (Dr. rer. nat.), © September 2014

"We may be small in numbers, but we stand for something bigger than anything the world can pin against us" –How to Train your Dragon

To my family and friends! Thanks for being there always. Love you all.

Dedicated to the loving memory of Yusra El-Zu`bi
1931 – 2004

ABSTRACT

Issue tracking is the process of reporting, assigning, prioritizing, reviewing, and resolving software issues. This process is crucial for successful collaboration between developers and for managing the projects' progress. However, in scientific software projects there is only little evidence on how issue tracking is practiced. Generally, scientific developers tend to informally handle and track these software issues, which becomes difficult when managing frequently evolving large-scale projects.

The goal of this dissertation is to support scientific software developers in their collaboration activities involving tracking and handling of software issues. The contribution of this dissertation is twofold. First, we describe a study on issue tracking in the domain of scientific software, in which scientists intensively collaborate to develop and use complex software systems. We analyzed and compared issue tracking data of two scientific and two open source software engineering projects. We also surveyed 612 project participants about their issue tracking practices, preferences, and problems. We found that members within software engineering projects tend to get involved in multiple issue tracking activities. On the other hand, members of scientific projects tend to be involved only in a single issue tracking activity. The results also indicate that issue tracking quality documentation is higher in the studied software engineering projects. These differences in issue tracking documentation quality and interconnectivity between activities indicate that issue tracking tools might need to be tailored to the specific domain needs of the individual projects. For example, issue tracking systems in scientific projects should provide additional features for administering the existing issue tracking activities, to provide an incentive to project members to exert more efforts.

Second, we describe *INExPERT*, a technique for issue assignees recommendation, which utilizes issue-tracking activities in identifying and ranking suitable assignees. *INExPERT* encourages the collaboration among project members by making it easier for software issues to be allocated to a specific project member, while at the same time hiding the technical and managerial complexities of the assignment activity. Our results indicate that *INExPERT* was able to have one or more suitable assignees in 88 % of the recommendations. In addition, an experiment with senior developers and managers in a large-scale scientific software project confirms the precision of *INExPERT* and its suitability for the scientific software domain.

ACKNOWLEDGMENTS

I would like to express my special appreciation and thanks to my advisor Professor Bernd Brügge, Ph.D. Thank you for encouraging my research and for allowing me to grow as a research scientist, as well as a person.

I am very grateful to Prof. Dr. Walid Maleej for having been a tremendous mentor for me. Thank you for helping and contributing a great deal to my research work, as well as my professional career.

I would also like to thank my committee members, Prof. Dr. Michael Gerndt and Prof. Dr. Michael Bader for taking part in my dissertation effort.

Special thanks to Helmut Naughton, Jonas Helming, Miriam Schmidberger and Maximilian Kögel for the great support all throughout my Ph.D. thesis.

Many thanks to all of whom supported me in collecting the data for my Ph.D. thesis, in particular, Dr. Benedikt Hegner, Victor Diez and Alex Hodgkins form SFT Group at CERN, Thomas Kuhr from Karlsruhe Institute of Technology (KIT), Stefan Kluth and Stefan Stonjek from the MPI group for Physik in Munich, Germany.

In addition, I would like to thank Oliver Meister, Tariq Saeed, Dina Helal, Shady Hussien, Omar Aly, Omar Othman, Ameirah Abu-azama, Mariam Rady, Ahmed Khalaf, Magda Hassan,Yomna Ali, Ahmed Farag, Ibrahim Tawifik, Ibrahim Sadik, Sara Mustafa, Salma Mahmoud, Ayman Khattab, Alyaa Mahmoud, Marouane Sayih, Karim Emara, Safey Halim, Daniel von Teller, Sherif Zaidan, Mohamed F. Eid and Wessam Abdrabo for their technical advice and support.

Thanks is due to the members of the chair of applied software engineering at TUM, in particular Jan Knobloch, Dennis Pagano, Tobias Roehm, Stefan Nosovic, Juan Haladjian, Monika Markl and Helma Schneider. Special thanks goes to the best support system and advisors in the whole world Emitza Guzman, Nitesh Narayan, Yang Li, and Han Xu. I really can not repay you for what you have contributed to my research and my personal life, you are my superheroes, love you guys.

A special thanks to my family. Words cannot express how grateful I am to my father, mother, brother, uncle, auntie Reem Kelani and Lubna Kelani for all of the sacrifices that you've made on my behalf. Your prayer for me was what sustained me this far.

I am also grateful to my friends Ghadeer Eresha, Doaa Nassar, Samah Shams Eldeen, Serena Fritsch, Rania Elhelw, Azza Fayek, Sara El-Tonsy, Sherine Hassab, Weaam El-Desouki, Salma Hassan, G. Weber, Giulia Maesaka, Mirna Ayman, Mai El-Sayad, Rana Salem, Nadiah El-Sayed, Yasmine Ogeil, Amr Nour El-deen, Haseeb Zia, Yujing Liu, Vinita Radhakrishnan, Hala El Ashi, Magi Mobasher, Tarek Attia, Hassan, Ahmed El-Atawy, Alaa Ibrek, Mohamed Abbas, Ahmed Azab, and Heba Khalifa for pushing me to strive towards my goal.

Finally, I would like to express my deepest gratitude towards God for all the blessings that he has bestowed upon me and for sending me the above mentioned people and for giving me the strength throughout the time.

CONTENTS

1 Introduction 1
- 1.1 Problem Statement 3
- 1.2 Research Approach 4
- 1.3 Thesis Structure 6

2 Related Work 7
- 2.1 Studying Issue tracking practices in Scientific Software projects 7
- 2.2 Assignee Recommendation 8

3 A Study on How Issue Tracking is Practiced within Scientific Software Projects 11
- 3.1 Study Design 11
  - 3.1.1 Studied Issue Tracking Activities 12
  - 3.1.2 Study Metrics 14
  - 3.1.3 Research Methods 14
  - 3.1.4 Case Studies 17
- 3.2 Study Results 20
  - 3.2.1 Data Analysis-Results 20
  - 3.2.2 Survey-Results 22
- 3.3 Summary 35
  - 3.3.1 Main Similarities 35
  - 3.3.2 Main Differences 35
  - 3.3.3 Specific Domain Issue Tracking Practicses 37

4 Issue Assignees Recommendation Technique-INExPERT 39
- 4.1 INExPERT Design 40
  - 4.1.1 Categorizing Issue Reports into Topics 41
  - 4.1.2 Issue-Tracking Activity Profile 42
  - 4.1.3 Assignee Recommendation 45
  - 4.1.4 Assignee Ranking 47
- 4.2 Case Study 48
  - 4.2.1 Formative Evaluation 48
  - 4.2.2 Pre-Post testing Quasi-Experiment 53
  - 4.2.3 Experiment Metrics and Findings 60
- 4.3 Evaluation 65
  - 4.3.1 Experts' Feedback 66
  - 4.3.2 Benchmarking 68

5 Conclusion and Future Work 75
- 5.1 Research Outcome Highlights 75
- 5.2 Research Limitations 76
- 5.3 Future work 76
- 5.4 Final Thoughts 76

I Appendix 77

A Appendix 79
- A.1 Layout of the Interview Used In INExPERT Formative Evaluation 79

A.2 A Copy of the Survey Used in Our Comparative Study 80

BIBLIOGRAPHY 85

LIST OF FIGURES

Figure 3.1 Study Design 12
Figure 3.2 Documentation Quality metrics 20
Figure 3.3 Frequency of issue tracking activities (Analyzed) 22
Figure 3.4 Participants' educational background 23
Figure 3.5 Participants' project roles 23
Figure 3.6 Participants' project experience 24
Figure 3.7 Participants' software engineering experience 24
Figure 3.8 Stated frequency of the issue tracking activities (survey) 26
Figure 3.9 Quality ratings of issue tracking practices 26
Figure 3.10 Benefit gained from using a bug tracking system (survey) 27
Figure 3.11 Tool preference Vs. performed issue tracking activity 28
Figure 3.12 Activities performed using a bug tracking system (survey) 28
Figure 3.13 Issue tracking problems (survey) 29
Figure 3.14 The significance of issue tracking problems (survey) 30
Figure 3.15 Issue tracking activities that are considered as an overhead 31
Figure 4.1 Issue Report's Topic Model 42
Figure 4.2 User's Activity Profile 43
Figure 4.3 Issue-Tracking Activities 43
Figure 4.4 Newly Reported Issue Topics Selection 46
Figure 4.5 Formative evaluation stages 49
Figure 4.6 Pre-INExPERT Stage's Activities 50
Figure 4.7 Post-INExPERT Stage's Activities 52
Figure 4.8 Comparison of a participant's decision before and after the use of INExPERT 53
Figure 4.9 INExPERT's Early Design Abstraction 55
Figure 4.10 Overview of the activities done to generate and rank a list of recommendations 55
Figure 4.11 Overview of INExPERT integrated into issue-tracking practices 58
Figure 4.12 Overview of the Website Architecture 59
Figure 4.13 Stage's Duration 62
Figure 4.14 Number of Assigned Issues (Manual Vs. INExPERT) 62
Figure 4.15 Number of Chosen Assignees Matching Recommendations (Manual Vs. INExPERT) 63
Figure 4.16 Number of Times a Participant Chose Same Assignee Within Pre- & Post-*INExPERT* Stages 63
Figure 4.17 Responses Vs. Qualitative Metrics 64
Figure 4.18 Experts' Judgement on Recommended Assignees Capabilities 66
Figure 4.19 Experts' Judgements Vs. Justifcations 67
Figure 4.20 Hit ratio for having the main resolver against the total number of issues 71
Figure 4.21 Top-n Hits, INExPERT vs LDA-SVM; Atlas 72

Figure 4.22 Top-n Hits, INExPERT vs LDA-SVM; Birt 72
Figure 4.23 Top-n Hits, INExPERT vs LDA-SVM; UNICASE 73
Figure 4.24 Overall hit ratio 73

LIST OF TABLES

Table 3.1 Heuristics used for identifying issue tracking activities 13
Table 3.2 Documentation Quality-Measures 15
Table 3.3 Survey-Measures 17
Table 3.4 Overview of studied projects 17
Table 3.5 Overview of original issue tracking data 19
Table 3.6 Overview of analyzed issue tracking data 19
Table 3.7 The survey samples 19
Table 3.8 Proportion test of Documentation Quality metric variables 21
Table 3.9 Number of open ended responses regarding issue tracking problems or overheads 32
Table 3.10 Issue tracking problem categories vs. number of open ended responses 32
Table 3.11 Issue tracking overheads categories vs. open ended responses percentage 33
Table 3.12 open ended responses regarding issue tracking enhancements 34
Table 3.13 Issue tracking enhancements categories vs. open ended responses percentage 34
Table 4.1 Heuristics Rules Used for Identifying Issue-Tracking Activities 44
Table 4.2 Experiment Participants 54
Table 4.3 Experiment Metrics 61
Table 4.4 Overview of the Investigated Projects 70
Table 4.5 Overview of Training and Testing Datasets 70
Table 4.6 Overview of Issue-Tracking Activities Distribution in Training Set 71

ACRONYMS

INEXPERT Issue Assignee Activity Profile Recommendation Technique

LHC Large Hadron Collider

CERN European Organization for Nuclear Research

VCS Version Control System

CASEA Creation Assistant for Easy Assignment

LDA Latent Dirichlet allocation

SVN Apache Subversion

PMAC Project Member Activity Counter

SCM Software Configuration Management

SVM Support Vector Machines

SMO Sequential Minimal Optimization

WEKA Waikato Environment for Knowledge Analysis

DREX Developer Recommendation with k-nearest-neighbor search and expertise ranking

1 INTRODUCTION

By the late 40s, the first generation of electronic computers had the capabilities to set up sequences of calculations, motivating scientists, especially in the domain of physics, to use them in developing their algorithms. This allowed them to achieve successful results in many applications such as ballistics and fluid mechanics. Currently, computer software has become an essential element for the scientific research community [10], marking the start of a community of "Scientific Software" developers. These developers create software that enables researchers to perform complex tasks such as the manipulating, analyzing, and modeling of large amounts of data with relative ease and speed when compared to manual methods. As the capabilities of computers' memory and speed is in constant increase, the use of computers and computer software has expanded within the scientific software community to include the development of large complex software systems. For example, 600 scientists collaborated in developing a large-scale software system that consists of 97 modules to analyze the products of high-energy collisions at ATLAS [1], a particle accelerator located at the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN), Europe's particle physics laboratory in Switzerland. The software system enables the consolidation of a large amount of raw data; its distribution on the Worldwide LHC Computing Grid; and, additionally, allows researchers to conduct scientific analysis, simulations, and reconstruction of their experiments.

The term "Scientific Software" according to Farhoodi et al. [21] refers to software that is developed by scientists for scientists, which is based on mathematical models or complex algorithms used to analyze, simulate, and even solve a specific scientific problem. According to Hannay et al. [29], scientists spend more time developing or using scientific software than they did a decade ago. This is due to the fact that computers nowadays have the ability to store and analyze large amounts of data, provide data visualization, and even perform 33.86 quadrillion floating point operations per second. Thus, scientists are able to address much more ambitious problems such as modeling genome structure, simulating the early evolution of the universe or even predicting the weather by analyzing past climate data.

Killcoyne et al. [42] consider scientific research to be a manic foraging exercise, that involves a constant change in the used techniques and research outcome. This is due to the scientific approach where scientists follow a hypothesis-based approach. This approach recognizes that there are multiple possible explanations for any given problem, in which all possibilities need to be investigated and multiple possible solutions have to be evaluated until the suitable one can be identified. However, scientists can reduce the time and effort investigating a given hypothesis through the progressive computational capabilities of computers and computer software. For example, assessing the complexity of a genome structure with a computer model and predicting how it will react under different environments; has now become possible. This is accom-

1 http://atlas.ch/

plished by scientists developing their own software using specialized algorithms and visualization techniques, as it requires a substantial amount of specific knowledge in the domain of bioengineering that is hard to achieve for many software developers.

Recently, software engineering has evolved with solution-oriented techniques such as continuous integration, issue tracking, unit testing, and agile methods to improve and support the development and quality of software systems. We cannot expect a scientific software developer to become an expert in all software engineering techniques, especially in a constantly evolving scientific research paradigm [78]. By the same argument, we cannot expect a software engineer to become an application domain expert in a specific scientific area. The fact that there is limited adoption of software engineering best practices within scientific software projects led to the creation of a gap between both communities [39].

Scientific software usually evolves through the combined collaborative efforts of different scientists over the years resulting in many software change requests that are referred to as software issues [69], which represent new requirements, software bugs, and features. Consequently, collaboration and issues management have become a challenge for scientists, particularly in large-scale projects. As Zaytsev et al. [86] pointed out that if the change in scientific software is not controllable, both the quality of software and the speed of development will suffer. Hence, it is important to provide developers with sufficient means to control technical and managerial complexities of change in scientific software development [47] .

Therefore, several software engineering researchers have collaborated with scientific software developers to investigate the extent to which basic software engineering practices. For example, Li et al. [46], Kelly et al. [41], and Zaytsev et al. [86] investigated the use of requirements reengineering, mutation testing, and continuous integration respectively in reducing the complexity and change in scientific software development. The applicability of these software engineering practices in scientific software development projects are influenced by many factors, such as developers' software engineering experience, targeted users behavior and team organizations [29, 58]. Mesh et al. [49] recommended that scientific software developers need to have a better understanding of these factors before attempting to apply or customize software engineering practices into their development process to guarantee their effectiveness [32, 80].

However, in scientific software projects there is only little evidence on how issue tracking is practiced. Killcoyne et al. [42] discovered that there is no direct support for collaboration among scientists on the development of software, since they tend not to implement any formal processes that are required for successful collaboration. Consequently, documentation related to planning, project management, and software implementation (such as defects, designs and potential improvements) are scarce[61]. In such a flexible environment, these highly independent developers themselves seem to reject any level of project management or instituting development processes, getting involved only in what interests them and serves their research activities [47]. The lack of collaboration support within scientific software projects adds several problems, especially in large-scale projects. This can have unfavorable outcomes, in particular when handling software issues related to software bugs. While software bugs in commercial software projects can be detected or prevented at early stages of development, in scientific software projects they have been shown to be discovered at the latest stages of

development, causing the retraction of a lot of published scientific work [49]. For example, a simple mistake such as flipping two columns of data in molecular structure analysis software has caused a group of researchers to retract three science papers and report that two papers in other journals also contained erroneous structures [50].

The main goal of this dissertation is to provide support to scientific software developers in their collaboration activities involving tracking and handling of software issues. Our work focuses on issue tracking activities related only to bug reports and feature requests, due to their significant impact on software quality. The contribution of this dissertation is twofold. First, we present an empirical study that analyzes the issue tracking practices of two scientific software projects and compare the outcome against two open source software engineering projects. The study can help software engineers understand how various members within different project domains collaborate on handling and tracking software issues. This is done through identifying general characteristics and points of strengths and weaknesses within the practiced issue tracking activities. Hence, the study outcomes can be exploited for collaboration enhancement, especially to overcome the limited support of issue tracking activities in the scientific software domain. Second, we present *'Issue Assignee Activity Profile Recommendation Technique (INExPERT)'*, a lightweight technique for the recommendation of software issue assignees that utilizes the project members' issue-tracking activities for identifying their roles, expertise, and involvement within the project. *INExPERT* aims at encouraging the collaboration among project members by making it easier for bug fixes and feature requests to be allocated to a specific project member while at the same time hiding the technical and managerial complexities of the assigning activity.

Parts of the presented contributions were published in [53, 54].

1.1 PROBLEM STATEMENT

Project members collaborate to keep track, coordinate, and handle ongoing software issues such as development tasks, new feature requests and bug fixes. This will help them avoid duplicated work, as each members' responsibilities are defined and visible to everyone in the project. Additionally, tracking software issues facilitate communication among project members as both developers and users are informed about the status of their issues and concerns.

Carver et al. [13] found that software teams in scientific software projects infrequently apply software engineering issue tracking practices. They informally track issues without defining the specific activities that must be implemented in order for them to successfully collaborate on fixing and handling the raised issues. As reported by Heaton et al. [30], they tend to keep track of issues in ad-hoc manner via mailing lists or their own personal logs. Consequently, the success of the collaboration is bound to the amount of effort spent by team members.

One of the main reasons why formal issue tracking is not considered important, is that scientists developing the software tend to give higher priority to achieving accurate scientific results over improving the software development process [15, 16, 72]. Additionally, the duration of scientific software projects is either too long or too short [14] which makes it difficult to commit to improving the project management practices. Another reason is that scientists developing the software are not familiar with many

software engineering practices such as: unit testing, issue tracking, code reviews, agile methods, version control, and software process management [13, 30]. According to Hanny et al. [29] and Prabhu et al. [63], 90% or more of the scientists developing software are self-taught or taught by other fellow scientists. Consequently, the lack of knowledge regarding a specific practice makes it harder for the software development team to adopt it [13]. Software development teams in scientific software projects are faced with the following two problems:

1. **Managing the complexity of collaboration in issue tracking**. In scientific software projects issues are not tracked through a set of defined activities that can guarantee the integrity and quality of information that describes how they were handled, managed, and resolved. Consequently, issues might end up being incomplete, conflicting, or even unrecorded, making it more expensive to handle and resolve them [30]. As stated by Faulk et al. [22], the dominant barriers to project productivity improvement are in the software processes. Therefore, it is important to provide managers of scientific software projects with means to control managerial complexities that can help them to improve issue tracking process. For example, we can provide them with a higher level of abstraction of the issue tracking process' variables such as the level of commitment, effort, common procedures used, and behaviors of involved project members. These abstractions can help managers easily control and improve the collaboration when tracking software issues. Additionally, we can automate the tedious parts of the issue tracking process that are error prone to help increase the process effectiveness.

2. **Assigning issue reports to developers can be error-prone**. An important step in getting an issue resolved is to assign the issue report to a suitable developer that will have enough expertise to resolve it. The correctness of that decision may have an important impact on the cost of the project, as it can increase the time taken to fix the software issue [75]. Nevertheless, in scientific software projects developers seem to reject any level of project management getting only involved in what interests them and serves their research activities [47]. As a consequence, issue assignment tends to be a voluntary action that is performed by project members who have the time to delegate ongoing issues to other team members to resolve them. Volunteers' experience, regarding how to assign issue reports may vary, because this requires contextual information about the project. Therefore, less experienced or new project members might not be able to assign bug reports correctly. As a result, manual issue assignment requires more effort and time to ensure that it was appointed to a suitable assignee.

1.2 RESEARCH APPROACH

This dissertation focuses on providing scientific software developers and managers with a mean to control the managerial complexities of issue tracking practices, by applying a set of software engineering strategies. As mentioned by Bryant et al. [11], a tighter coupling between the description of a software system and its application domain gives a great potential to improve its correctness and reliability. Furthermore, this

leads to more opportunities for software automation. To that end, we applied two software strategies; that were mentioned by Faulk et al. [22]: measurement and automation. In order for the currently implemented issue tracking practices to be effective, we have to continuously observe and measure how scientific software projects members perceive them and to which extent they are influenced by them. Additionally, tedious tasks within issue tracking practices (such as issue assignment) can be improved if the repetitive work belonging to it is automated. To this end, we first investigated the current state of adopted issue tracking practices within scientific software projects by exploring the question, *How do software developers track issues in scientific software projects and how can we make the process more effective ?* In order to address this question, we first reviewed the existing literature related to the characteristics and problems of collaboration software engineering practices in the scientific domain, focusing on issue tracking activities. The literature review helped us gather the basic foundation that identifies the characterizations of issue tracking practices in scientific software projects; along with potential strategies that could make issue tracking more effective. Second, we conducted an empirical study that provides a deeper insight into the level of commitment, effort, used common procedures and behaviors of project members involved in tracking issues. The empirical study included multiple case studies from both the software engineering and scientific software projects. It was designed to help software engineers determine the necessary improvements for issue tracking practices in both project domains. The study investigates three main issue tracking activities: reporting, management, and resolvement of software issues. For each studied activity we defined five metrics : *interconnectivity, tools, documentation quality, frequency, and enhancements and problems*. These were measured using quantitative and qualitative values that reflected: (1) the inconsistency or incompleteness of the tracked information, (2) the tools used in tracking each activity, and (3) the problems or enhancements related to it, which helped us identify:

- *The strength of interconnectivity between different issue tracking activities;*
- *The tools that are frequently used in performing a certain issue tracking activity;*
- *Inconsistency or incompleteness within information related to the recording of an issue tracking activity;*
- *The significance of a specific issue tracking activity in terms of its frequency compared to others;*
- *The problems encountered during each issue tracking activity;*
- *The types of enhancements needed to make issue tracking practices more effective.*

Third, we provided a technique that supports the task of issue assignment by identifying and recommending suitable assignees, i.e., developers, to submitted software issue reports. The technique enabled us to:

- *Identify a suitable assignee to resolve a specific issue report;*
- *Rank the recommended assignees according to their expertise and roles.*

The issue assignees recommendation and ranking technique, *INExPERT*, is based on identifying and ranking suitable assignees through mining their profiles. Profiles are defined by roles and expertise mined from the history of all the issue tracking activities, i.e., review, assign, and resolve.

Finally, we evaluated the precision and suitability of *INExPERT* within scientific software projects, by implementing a prototype. Using the *INExPERT* prototype, we conducted a formative evaluation method that involved two iterations of direct feedback from domain experts regarding the points of improvement within *INExPERT*. The feedback was obtained using a quasi-experiment and semi-structured interviews. Afterwards, the feedback was analyzed, and changes were applied to the *INExPERT* prototype to enhance its precision and make it more suitable for use in scientific software projects. Then, a second iteration of feedback was obtained to evaluate the applied enhancements.

1.3 THESIS STRUCTURE

The dissertation is organized as follows:

Chapter 2 presents the related work on software engineering concepts directed to the development of scientific software. We focus on two aspects: issue tracking practices and issue assignee recommendation techniques.

Chapter 3 describes the empirical study we conducted to help characterize and determine areas of improvement within issue tracking practices in the scientific software domain. The chapter includes details of the study design, the used case studies, the outcomes of the case studies and their implications.

Chapter 4 we present our issue assignees recommendation technique, *INExPERT*. The chapter includes the steps used for the categorization of topics, creation of activity profiles for potential assignees, assignee recommendation for new issue report, and assignee ranking. It also provides: (1) an overview of how we evaluated the suitability and precision of *INExPERT* in scientific software projects, and (2) details of the evaluation setup, projects involved, and evaluation outcome and its implications.

Chapter 5 summarizes the highlights and implications of the outcomes of our research work, as well as details about the limitations of the empirical study we conducted and the issue assignees recommendation technique we implemented. We conclude our work with remarks on suggestions related to future directions for research.

2 RELATED WORK

In this chapter we give an overview of the relevant existing research contributions that tie in with the work presented in this dissertation. Section 2.1 describes the existing empirical studies that use issue tracking information in analyzing the software development process within several projects, as well as empirical studies focusing on software engineering concepts directed to the development of scientific software. Section 2.2 describes the approaches that semi-automatically assign different types of artifacts to developers. We discuss related work employing artifacts similar to and different from ours. For empirical research effort we considered different goals such as evaluating process quality and identifying the effect of different development paradigms on the adopted software engineering practices. For the assignee recommendation approaches, we considered various other factors, such as developer preferences and workloads.

2.1 STUDYING ISSUE TRACKING PRACTICES IN SCIENTIFIC SOFTWARE PROJECTS

Issue tracking systems are an important source for gaining insight on the software quality and the reproduction of software issues [76]. Therefore, most empirical studies tended to analyze groups of widely used software projects with a long data history of issue tracker's database [33]. Commonly, these studies investigated the software development process. However, they tended to focus on different goals. Bettenburg et al. [9] investigated the features that makes a bug report more informative and useful. Bachmann et al. [4] provided a group of measures that evaluate the software development process's quality and characteristics and their affects on the number of bugs . Saha et al. [67] analyzed the proportion, severity, assignment, reasons, and the nature of fixes related to persistent bugs within four open-source projects to understand the extent of and reasons for their existence. Cavalcanti et al. [17] surveyed 36 participants to investigate the effort invested in assigning change requests to appropriate developers. They identified fundamental strategies to perform the assignments and the complexity involved in them. Other studies [89, 37, 45, 48] focused on improving the issue tracking system for the purpose of improving the issue modification process. Additionally, other studies [85, 60] investigated the differences between software projects having different development paradigms.

During recent years, software for scientific computing and research purposes has received increased attention from the software engineering community [58]. For instance, Carver et al. [12, 6, 13] and Heaton et al. [30] both investigated the use of software engineering practices such as issue and bug tracking, integration testing, regression testing, and code reviews in the scientific software community by conducting interviews and surveys with domain experts. Hanny et al. [29] conducted an online survey and received almost 2000 responses shedding light on how scientists develop and use scientific software applications. Prabhu et al. [63] randomly selected 114 researchers from diverse fields of science and conducted a survey to investigate the

software engineering practices and tools applied within the participants' community. Sanders [69] et al. interviewed 16 scientists to identify the approaches they used to manage risks in the development process, testing, and documentation. Generally, the above mentioned studies were able to identify a generalized perception of software engineering concepts and their level of usage within the scientific community; no detailed information was obtained on software issue tracking tools and approaches.

Pawlik et al. [61] conducted a study that included 21 interviews and thematic analysis of interview data. They aimed at identifying the perception of the scientific software community on which documentation approaches and tools are most useful to them. Additionally, they investigated the influence of the scientific software user community on the documentation practices. However, they did not focus on documentation practices related to implementation defects, i.e., software issues.

Many researchers such as Segal et al. [73, 74], Howison et al. [36], and Kelly [40] conducted literature reviews, interviews, and data analysis to gain different insights on the culture of how and why scientists develop software and cooperate among different scientific disciplines and domains. Monteith et al. [52] used several state-of-the-art analysis techniques to examine the artifacts from several scientific research projects to identify problems inhibiting sustainability in the scientific software development, maintenance, funding, and leadership.

To our knowledge, none of the above mentioned studies focused on mining issue tracking repositories to study the issue tracking practices within the scientific software domain. In general, these studies did not present any clear indications on the quality of documentation, common implemented procedures, and used tools by scientists in the tracking and management of software issues. Therefore, it is our aim to gain a deeper insight on the level of commitment, efforts, used common procedures, problems encountered, and behaviors of scientists involved in tracking issues. Some of the metrics we used in our study were based on the work of Bachmann et al. [5], which evaluated the quality of software process data obtained from several artifacts including issue trackers.

2.2 ASSIGNEE RECOMMENDATION

Most of the existing work for assignee recommendation relied on analyzing the bug reports and other artifacts stored in the same repository–for example, comments on the bug reports and the attached stack trace file [87]. Several other approaches considered heterogeneous artifacts collected from different repositories used within a software project, eg., Version Control System (VCS) commit history and source code artifacts [51, 70, 24], to determine the expertise of the developers. Work from Yingbo et al. [84] used event logs of the workflow system to identify patterns of activities, which is common with our work. However, they used machine learning to identify the various activities each person undertakes, and focus on the field of workflow.

Anvik et al. [2] applied a supervised machine learning algorithm in their work on the information extracted from the bug repositories of three big open source software projects. They achieved high precision on the Eclipse and Firefox development projects. They also highlighted an important project-specific aspect, i.e., "one developer dominating the report resolution process", which we try to overcome in our approach by

allocating appropriate weights to various other activities. Anvik et al in 2013 extended their work by developing 'Creation Assistant for Easy Assignment (CASEA)', a tool that uses a project member's knowledge to create an assignment recommender specific to the software development project. It assists the user in labeling and filtering the bug reports used for creating a project-specific assignment recommender. The authors claim their work is more practical compared to our approach.

Cubranic et al. [18] emploed text categorization using a naive Bayes classifier to predict the developer that should work on the bug based on the bug's description. They predict 30% of the assignments over a collection of 15,859 bug reports from the Eclipse bug tracking repository. Hossen et al. [35] developed 'iMacPro', for assigning the incoming change requests to appropriate developers who have the necessary expertise to resolve them. iMacPro uses Latent Semantics Indexing to retrieve textual description of an incoming change request and locate it to relevant units from a source-code snapshot, which are change prone. To that end, authors and maintainers of change-prone source code units are most likely to best assist with the incoming change request's resolution. A similar approach by Shokripour et al.[75] predicted the source code files that will be changed to fix a new bug report, by extracting from four distinct text information sources indexes of unigram noun terms that links it to source code files. Afterwards, they used these predicted source code files to recommend suitable developers for handling the new reported bug.

Wu et al. [81] presented an approach called Developer Recommendation with k-nearest-neighbor search and expertise ranking (DREX) that performs developers recommendation, using K-Nearest-Neighbor search that is based on: (1) bug similarity and (2) social network expertise ranking metrics. Work from Zhang et al. [87] also considered social networks for retrieving candidate developers, along with creating a concept profile for extracting bug concepts and topic terms. On the other hand, Yang et al. [83] constructed a multi-developer network that helps in verifying the developers' ability to fix a given bug report based on the number of received comments and sent commits. This work not only considered the comment activities, but also focused on the commit messages for source code files change. Work from Mainur et al. [64] additionally considered developers' workloads to estimate the required time to fix a new bug. They applied different variants of Greedy Search (with varying parameters) for nine milestones of the Eclipse JDT project. Similarly, work from Park et al. [59] also considered the workload factor. They applied Latent Dirichlet allocation (LDA) to identify bug types, and quantify each value of the developer's profile as the average time to fix the bug in the corresponding type. Baysal et al. [8] presented a theoretic framework for automated bug assignment considering developer preferences, expertise, and workloads.

Nagwani et al. [55] implemented a tool that ranks developers within a certain project based on their contribution within bug resolution and fixes. The contribution score of each developer is calculated using a number of attributes from the bug tracking repository that included; bug fixes of a certain severity and priority, and reporting new problems and comments, i.e., bug resolving activities. Gousios et al. [25] proposed a model for calculating the activities a developer has performed during the development process, which they refer to as "contribution" towards different tools like Apache Subversion (SVN), bug tracking repository, wiki, mailing lists, and forums. Some ex-

ample, of these activities are, adding a line of code, committing fixes to code style, committing code that closes a bug, closing a bug report, reporting a bug, commenting on a bug report, and starting a new wiki page. Xie et al [82] presented a bug assignment recommendation model that is based on topic models and bug resolving activities. Upon the arrival of a new bug report, a ranked list of developers is generated. These developers are likely to contribute to resolving the new bug according to their matching interests and expertise. Nguyen et al. [57] implemented FixTime, a time-aware approach for issue assignment. FixTime is a topic-based, log-normal regression model to predict the resolution time of a given issue if it is assigned to a given developer. FixTime uses that model to predict all available developers and ranks them based on the predicted resolution time to recommend the fixing assignment. The approach we used in ranking the developers differs from the work of Nagwani et al. and Xie et al. in: (1) The activities they considered are contribution and resolving activities, while we focus on resolving, assigning, and reviewing. (2) The attributes of the bug tracking repositories they used in identifying the activities are: bug report's priority, status, and comments, while we use the bug report's status and resolution.

3

A STUDY ON HOW ISSUE TRACKING IS PRACTICED WITHIN SCIENTIFIC SOFTWARE PROJECTS

Issue tracking is the process of reporting, assigning, prioritizing, reviewing, and resolving software issues. In large complex projects, issue tracking is crucial for a successful collaboration between developers and for managing the projects' progress. Previous research [67, 89, 37, 45, 48] has focused on issue tracking in large Open Source projects, often carried-out by developers with substantial software engineering background. However, there is only a little evidence on how issue tracking is practiced in other domains.

In this chapter, we studied issue tracking in the domain of scientific software development, in which scientists intensively collaborate to develop and use complex software systems. We empirically analyzed and compared issue tracking data of two scientific and two open source software engineering projects. We also surveyed 612 project participants about their issue tracking practices, preferences, and problems.
Chapter content:

- An empirical study that utilizes activities within issue tracking repositories, to identify project specific overheads and problems within issue tracking practices applied in software projects.
- A comparison between the results obtained from the scientific software projects and the open source software engineering projects.
- An insight into the pattern of activities performed by developers in issue tracking repositories of various project domains. These patterns indicate how various project members interact with issue reports and how issue tracking is practiced within each project domain.

The rest of the chapter is organized as follows. Section 3.1 describes the design of our study including the research questions, methods, and data used. Then, we report on the results of the survey and the data analysis in Section 3.2. In Section 3.3 we summarize our findings, and discuss their limitations and implications for researchers and practitioners.

3.1 STUDY DESIGN

In this section we describe how the study was designed to determine the characteristics of issue tracking activities, behaviors and impediments belonging to a specific project. In the study we targeted three main issue tracking activities: reporting, management and resolvement of issue reports. As shown in Figure 3.1, for each issue tracking activity we focused on identifying five metrics: *interconnectivity, tools, documentation quality, frequency, enhancements, and problems*. These metrics were used to answer the following research questions:

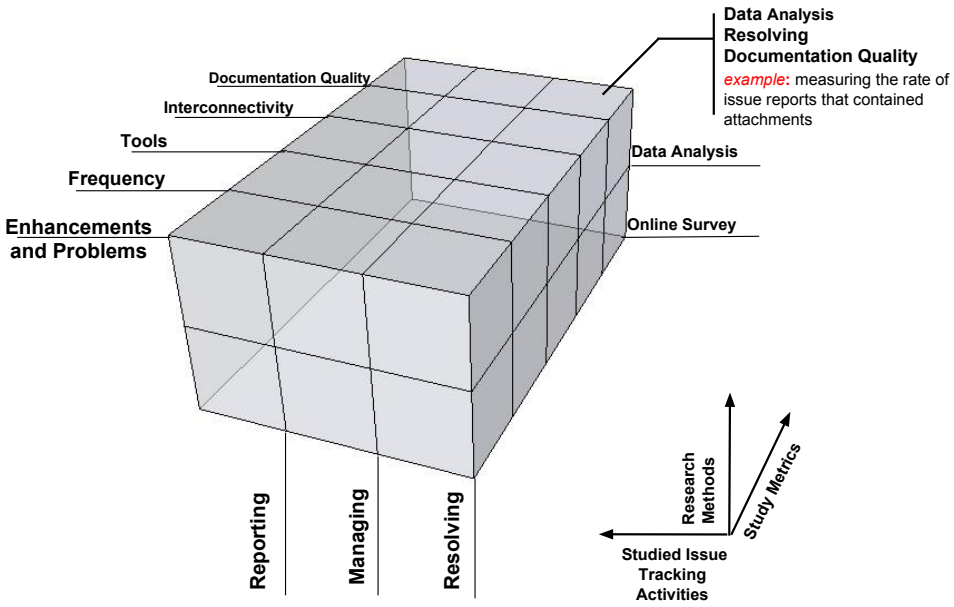

Figure 3.1: Study Design

- *Are issue tracking activities interconnected with each other?*
- *Which tools are frequently used in performing a certain issue tracking activity?*
- *Is the documentation of issue tracking activities inconsistent or incomplete?*
- *Are there any activities more frequently performed compared to others?*
- *What are the problems encountered during each issue tracking activity?*
- *What types of enhancements are needed to make issue tracking practices more effective?*

These metrics provide a deeper insight on the level of commitment, effort, common procedures used, and behaviors of project members involved in tracking issues. Additionally, pointing out the problems encountered during issue tracking activities, their frequencies, and their impacts. This allows the identification of the needs of the project members, their priorities, and potentials for improvement.

3.1.1 *Studied Issue Tracking Activities*

Within a typical software project, an issue report goes through a standard lifecycle, in which it gets submitted, assigned, resolved, verified and finally closed. The issue tracking activities are actions done to transfer an issue report from one of these states into another. Considering our goal in providing distinctive characterization of the issue tracking practices within a certain project, we categorized the characteristics of issue tracking practices into three main activities: reporting, managing, and resolving. Reporting includes the capturing and communication of a discovered issue. Managing an issue report is the activity of deciding the progress of the issue within its life cycle, which includes assigning, prioritizing, and reviewing the issue. Finally, resolving concerns how the issue's resolution was implemented.

The way each of these activities are represented within different issue tracking repositories may vary due to the fact that issue report attributes may differ from an issue tracking repository to another. For example, in a certain issue tracking repository, two attributes (namely: status and resolution) are used to give a more descriptive trace to the report's status, (e.g. an issue can have a "Resolved" *status* together with a "Fixed" *resolution*), while in another issue tracking repository only one attribute (namely: *status*) is used to give a brief indication of the report's current status.

We identify an issue tracking activity as a specific pattern in the history logs of the issue tracking repository. The pattern is a series of specific changes within certain issue reports' attributes. For example, an assigning activity is *acknowledged* if the history log indicates that the report's attribute referring to the *assignee* (the person involved in resolving the issue) has been updated with a valid assignee id. We have formulated a set of heuristics summarized in Table 3.1 that defines the patterns of history log entries. The heuristics can be further adjusted to fit the issue reports' attributes of a certain issue tracking repository.

Table 3.1: Heuristics used for identifying issue tracking activities

Activity	**Rule**
Reporting	If the history log recorded the submission of an issue report. Then, the individual responsible for that submission, i.e. (issue report submitter) has performed a reporting activity.
Assigning	If the history log recorded a change in the issue report's attribute referring to the person involved in resolving the issue, i.e. assignee. Then, the individual responsible for that change has performed an assigning activity. (i.e. has assigned/reassigned an issue report to an existing assignee).
Prioritizing	If the history log recorded a change in the issue report's attribute referring to the issue report's priority. Then, the individual responsible for that change has performed a prioritizing activity. (i.e. has set the priority of an issue report to certain level).
Reviewing	For an *unassigned* issue report, if the history log recorded a change in the issue report's attribute referring to its resolution status, indicating that the issue report was either invalid or a duplicate. Then, the individual responsible for that change has performed a reviewing activity. For a *resolved* issue report, if the history log recorded a change in the issue report's attribute referring to its status, indicating that the issue report's resolution was set toverified. Then, the individual responsible for that change has performed a reviewing activity. For a *closed* issue report, if the history log recorded a change in the issue report's attribute referring to its status, indicating that the issue report was reopened (i.e. status changed from closed to open). Then, the individual responsible that change has performed a reviewing activity.
Resolving	For an *already assigned* issue report, if the history log recorded a change done by the issue report's assignee to the issue report's attribute referring to its resolution status indicating that a formal resolution has been reached (e.g. fixed/won't fix/duplicate). Then, the individual responsible for that change has performed a resolving activity.

3.1.2 *Study Metrics*

This section presents a brief description of the five metrics used to characterize each studied issue tracking activity.

Interconnectivity: is a metric that measures the level of interdependence between two issue tracking activities, indicating the influence of one activity on the other. This provides an estimate on the project members dynamics and how defined are the boundaries between the issue tracking activities. In other words, the metric of interconnectivity can indicate if the studied projects have distinct subgroups performing only one specific activity or if they consist of subgroups that perform multiple activities belonging to other groups.

Tools: is a metric that determines the frequently used tools in performing a certain issue tracking activity, indicating the common procedures in performing a certain activity.

Documentation Quality: is a metric that determines how well a certain issue tracking activity was documented. This provides an indication on the performed activity conformance with issue tracking specifications. In addition, it verifies the completeness and consistency of information related to the documentation of the performed activity.

Frequency: is a metric that evaluates the importance of a certain activity in terms of its frequency compared to others, enabling to identify irregularities or inconsistencies within the issue reports' lifecycle.

Enhancements and Problems: is a metric that determines the problems encountered during each issue tracking activity, pointing out the areas, which needs to be improved. Additionally, it defines the types of enhancements needed by the project members that can add significant value to the project.

All data used in calculating these study metrics were collected using both research methods: (1) data analysis of project artifacts and (2) online-survey, are further discussed in the next section.

3.1.3 *Research Methods*

This section presents details of the research methods used in collecting quantitative and qualitative variables that represent the study metrics.

3.1.3.1 *Data Analysis-Measures*

We used data analysis as a method for collecting quantitative values to determine the *Documentation Quality*, *Frequency*, and *Interconnectivity* metrics. We studied the contents of the issue tracker artifacts: feature requests and bug reports, focusing on the changes in the history logs concerning issue reports' attributes such as status and resolution. The measured values were used to identify three main aspects: (1) if a certain issue tracking activity was performed according to a predefined specification, (2) relationships between the issue tracking activities performed by project members, and (3) the frequency of a certain activity with respect to the total number of performed issue tracking activities. This was mainly done by counting the issue reports having

Table 3.2: Documentation Quality-Measures

Activity	Measure	Description	Incident Type
Reporting	RCNI	*Rate* of issue reports that contained all the necessary attributes for describing the problem. (*title, description, operating system, release and system component*)	Positive
	RCA	*Rate* of issue reports that contained attachments.	Positive
Assigning	RA	*Rate* of assigned issue reports. (closed reports)	Positive
	ATC	*Assignee tossing count*, represents the number of times an issue report was reassigned.	Negative
Reviewing	RNR	*Rate* of issue reports that where closed but had no resolution, i.e. the issue reports attribute indicating its resolution was set to null or indefinite value.	Negative
	RRCSP	*Rate* of issue reports that where closed and resolved by the same person.	Negative
Prioritizing	PCC	*Priority change count*, represents the number of issue reports that encountered changes in the attribute indicating its priority.	Positive
Resolving	RNRA	*Rate* of issue reports that were not resolved by the assignee.	Negative
	RCR	*Rate* of issue reports that had comments indicating a description of how it was resolved.	Positive

its contents (i.e. attributes) meeting a certain criteria. For example, we counted the number of issue reports that were having status: closed, but did not have any trace on how it was resolved, which can indicate that the actual method of performing the resolution activity was not documented, i.e., the issue report attribute referring to its resolution has no actual trace to how the problem was solved (equals to null). In this case, this measure can be used to represent the *Documentation Quality* metric.

The documentation quality measured values are divided into two categories: negative and positive incidents. The negative incident concerns an issue tracking activities that were incompletely or inconsistently recorded or an incident that indicates a low level of collaboration or unclear information or lack of experience. While a positive incident concerns the activities that were performed according to specific requirements that guarantees the confidence of recorded information or high level of collaboration or useful information or high level of experience. Our interpretation of a positive and a negative incident was deduced from the work of Bachmann et al. [5]. Consequently, both types of incidents can give an estimate on the maturity level of the issue tracking process within the investigated projects, i.e., the project with higher number of positive incidents and less number of negative incidents implies a high maturity level of its issue tracking process. Table 3.2 gives a short description of each measured value representing the *Documentation Quality* assessment metric. The measured values were then categorized based on the concerned activity and incident type.

To determine how activities interconnect with each other, we calculated Project Member Activity Counter (PMAC), that represents for each project member$_i$ the num-

ber of times he performed a certain issue tracking activity$_j$. After then we calculated Pearson Product-Moment Correlation Coefficient r [68], between all possible combination of every activity pair using the P_iMA_jC variable. As shown in equation 3.1, we determined the interconnectivity between two activities X and Y by calculating the correlation coefficient r of P_iMA_jC variables belonging to each project member$_i$ that is related to activity X and Y. For interpreting the relationships among P_iMCA_X & P_iMCA_Y, we used the scale provided by Salkin [68].

$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{P_iMCA_X - \overline{P_iMCA_X}}{S_{P_iMCA_X}}\right) \times \left(\frac{P_iMCA_Y - \overline{P_iMCA_Y}}{S_{P_iMCA_Y}}\right) \tag{3.1}$$

- n is the number of project members.
- The mean of the P_iMCA_X & P_iMCA_Y is denoted by a horizontal bar over it.
- The standard deviation of the P_iMCA_X & P_iMCA_Y is denoted by $S_{P_iMCA_X}$&$S_{P_iMCA_Y}$.

We calculated the frequency f of an issue tracking activity i by counting the number of times the activity i has occurred within a specific period of time as shown in Equation 3.2. Afterwards, we normalized the frequency through dividing it by the total number of issue tracking activities N.

$$f_i = \frac{n_i}{N} \tag{3.2}$$

- N represents the total number of issue tracking activities performed within a specific period of time.

3.1.3.2 *Survey-Measures*

We conducted an online survey to collect quantitative and qualitative data for a representative evaluation of our study metrics. The survey included 16 closed questions, which were divided into two groups: (1) twelve questions were designed to measure specific data values for determining one or more study metrics, and (2) four demography questions were designed to capture the survey participants' educational background, software engineering background, and project experience. The demography data can serve as a useful tool for drawing statistical analysis on the influence of the participants' environment and background on issue tracking activities, which if used can add another dimension to the study. However, this is not the focus of our research.

Table 3.3 gives a short description on each of 16 survey questions, categorizing them according to the study metric(s) they represent. A detailed copy of the 16 questions and provided answer options can be seen in the Appendix at Section A.2.

Table 3.3: Survey-Measures

Question ID	Question Content	Study Metric
Q1	Performed activities	Interconnectivity, Frequency
Q2	Average completion time per activity	Documentation Quality
Q3	Rating of project's issue tracking practices	Documentation Quality
Q4	Perceived benefits of using an issue tracker	Tools
Q5	Activities performed using an issue tracker	
Q6	knowledge level of issue tracking tools	
Q7	Preferred tools in each activity	
Q8	Frequently encountered problems	Enhancements & problems
Q9	Severity of encountered problems	
Q10	Activities that are seen as an overhead	
Q11	Project specific overheads & problems	
Q12	Project specific issue tracking enhancements	
Q13	Participant's role in the project	Demography
Q14	Participant's project experience	
Q15	Participant's educational background	
Q16	Participant's software engineer experience	

3.1.4 *Case Studies*

We applied our study metrics within several case studies, in which we empirically analyzed the issue tracking logs of two scientific software projects and conducting an online survey with their project team members. Additionally, we compared the outcomes against two open source software engineering projects. The investigated four software projects are introduced in Table 3.4. ATLAS-Reconstruction and Bell2 projects represent the scientific software domain, while Eclipse and UNICASE are projects from the software engineering domain, i.e., the nonscientific domain. The projects were selected to ensure diversity within the quality of issue reports, how they got assigned, reviewed, and resolved; so that we could draw unbiased interpretations. For each project we defined two different datasets that were used in the data analysis and online surveying, they are described in Section 3.1.4.1 and 3.1.4.2.

Table 3.4: Overview of studied projects

	Scientific Software		Software Engineering	
	ATLAS-Reco	**Belle2**	**Eclipse**	**UNICASE**
P. language	C++/Python	C++/Python	Java	Java
Lines of code	~7 Mio.	~1 Mio.	~3 Mio.	~5 Mio.
Issue tracker	Savannah	Redmine	Bugzilla	UNICASE
Avg. reports/ day	1	2	100	1
Management of issue reports	Volunteer-based	Volunteer-based	Developer-based	Developer-based

ATLAS-Reco[1] : ATLAS is a particle accelerator experiment conducted at the LHC at CERN in Geneva, Switzerland. ATLAS roughly involves 3,000 scientists and engineers from 165 institutions and 35 countries. The ATLAS software infrastructure consists of 97 sub-projects, which consolidate raw data and distribute it on the Worldwide LHC Computing Grid, providing scientific analysis, simulations, and reconstruction of different scientific experiments. We study the ATLAS-Reconstruction project which is responsible for the event reconstruction in the experiments. The ATLAS-Reconstruction team created about 7 Mio. lines of code. Managing of issue reports depends on the voluntarily efforts of team members, in which they use Savannah[2], a proprietary web-based tool since 2003.

Belle2[3]: is a particle physics experiment conducted by the Belle Collaboration, an international project of roughly 400 physicists and engineers from 19 countries and 65 institutions organization around the globe. It investigates CP-violation effects at the High Energy Accelerator Research Organization (KEK) in Tsukuba Japan. We studied the issue tracking of Belle2. It includes 28 stakeholders who created about 1. Mio lines of code. Managing of issue reports is based on the voluntarily efforts of the team members. Starting from 2011, they are using Redmine[4], an open source web-based issue tracking system.

Eclipse: is an open source development platform, which comprises extensible frameworks; tools, and runtime environments to build, deploy, and manage software systems. The Eclipse community is one of the largest open source communities including 11 million users and more than 30 000 individuals who are involved in the issue tracking process. Eclipse is frequently used in software engineering empirical studies [66], as its data is is publicly available and easily accessible from the web. For issue tracking since 2001, Eclipse uses Bugzilla[5], a popular open source bug tracker.

UNICASE[6]: is an open source CASE tool, which enables to collaboratively create and manage software engineering models such as class and use case diagrams. UNICASE also allows for linking the models and their elements to tasks, bug reports, and other project documentation. The UNICASE team involves 51 individuals who have created about 5 Mio lines of code. The team uses UNICASE itself as an issue tracker.

3.1.4.1 *Data Analysis-Dataset*

The dataset used in the data analysis method consisted of issue reports as well as the activity history (i.e. records of users' interactions with the bug tracker). We transformed the database dumps of the issue trackers into a unified database. The unification enabled us to (a) filter unnecessary details such as the user credentials, (b) unify the attribute name across the databases simplifying the referencing and comparison, and (c) conduct the statistical analysis more efficiently to avoid complicated join operations and attribute matching. The unified issue report data included the following information: issue_id, project_id, title, description, assignee, submitter, cre-

1 http://www.atlas.ch/
2 https://savannah.cern.ch/projects/atlas-bugs/
3 http://belle2.kek.jp/
4 http://www.redmine.org/
5 http://www.bugzilla.org/
6 https://code.google.com/p/unicase/wiki/UNICASEClientNavigation

ation_date, priority, resolution, status and category. The unified activity history included the following information bug_id, activity_id, project_id, activity_type, who, when, old_value, new_value. The data were filtered to ensure the accuracy of the analysis.

The analyzed sample consisted of only closed issues to ensure that the issue reports are at a steady state that is reflecting a complete life cycle. Thus, including all information about reporting, resolvement and management activities. Table 3.5 and 3.6 summarize the four original and analyzed datasets.

Table 3.5: Overview of original issue tracking data

	ATLAS-Reco	**Belle2**	**Eclipse**	**UNICASE**
DB size	47.9 MiB	960.0 KiB	2.9 GiB	1.7 MiB
Studied period	2003-2012	2011-2012	2001-2010	2008-2011
# reports	3035	566	316911	1079
# project members	501	29	30908	53

Table 3.6: Overview of analyzed issue tracking data

	ATLAS-Reco	**Belle2**	**Eclipse**	**UNICASE**
# closed issue reports	2923	364	66581	812
# project members	312	16	6818	37
# issue tracking activities	6525	415	234514	2330

3.1.4.2 *Survey-Dataset*

The survey sample included all individuals from the target population who have performed at least two issue tracking activities in one of the studied projects. To ensure a minimum level of involvement in the project and knowledge with its issue tracking practices we excluded individuals who participated only once in a single report. To be representative, our sample should reflect different stakeholders in the project. Therefore, we emailed all addresses available in the issue tracking repository from each population.

Table 3.7: The survey samples

	ATLAS-Reco	**Belle2**	**Eclipse**	**UNICASE**
Population	399	28	32178	51
# of full responses	72	15	515 *(240 Reporters & 275 Developers)*	10
% of sample	18	53.6	1.6	19.6
Margin of error	10.21	17.56	4.28	28.06

The sample size we targeted was calculated with an online Sample Size Calculator [7] with a 95% Confidence Level and 5% Confidence Interval (i.e. Error Margin). As, response rates of online surveys typically vary between 36% to 93% [19], we doubled the calculated target sample size to get enough responses. We then sent personalized

7 http://www.surveysystem.com/sscalc.htm

emails to the participants and reminded them after one week, two weeks, and one month. The details of the samples are summarized in Table 3.7. Since Eclipse had the largest number of target population among all the other projects, we wanted to increase the quality of its participants interactions to avoid excessive survey length and irrelevant questions, which eventually would help us achieve high number of responses within our sample. To that end, we have created two survey instances, one similar to the one used within all the other projects and another one that is shorter in length containing only questions relevant to issue reporters.

3.2 STUDY RESULTS

This section provides a summary of the outcome from applying the data analysis measures described in Section 3.1.3.1 on the issue tracking repositories belonging to the studied projects. Additionally, the section summarizes the outcome of analyzing the 612 survey responses based on the study metrics described in Section 3.1.3.2.

3.2.1 *Data Analysis-Results*

This section provides a summary of the outcome from applying the data analysis measures described in Section 3.1.3.1 on the dataset described in Section 3.1.4.1. The data analysis measures represent the study metrics: documentation quality, interconnectivity, and frequency. This provides an in-depth knowledge on the level of collaboration among project members, and the quality and amount of effort exerted in documenting the issue tracking activities.

3.2.1.1 *Documentation Quality*

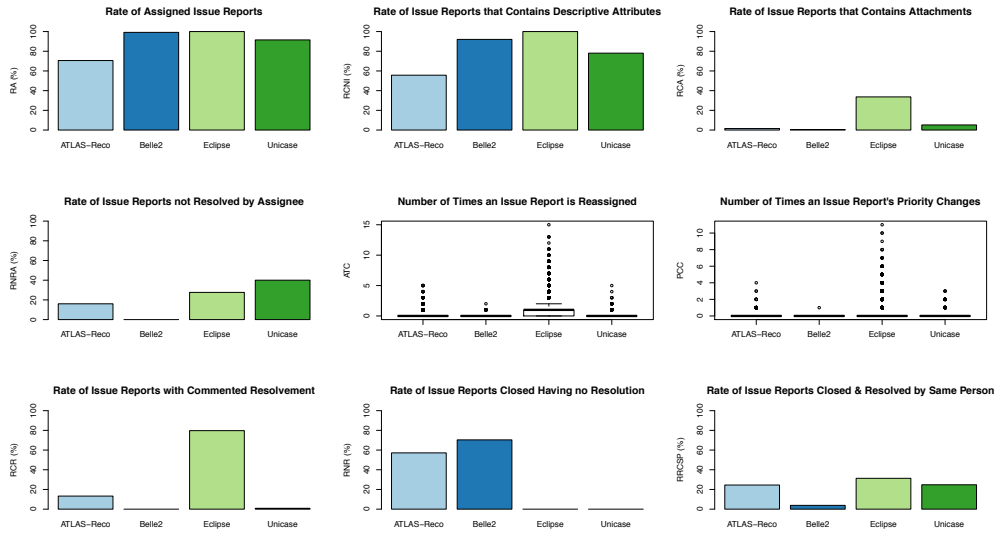

Figure 3.2: Documentation Quality metrics

Figure 3.2 shows, nine measures indicating the documentation quality of issue tracking activities within the four investigated projects. From, the rate of issue reports with commented resolvement **(RCR)** and the rate of issue reports that contains descriptive attributes **(RCNI)**, we can deduce the amount of effort exerted from the project members when they are resolving and assigning issue reports (i.e., positive incident). Additionally, given the rate of assigned issue reports **(RA)** and the rate of closed issue reports having no documentation of its resolution **(RNR)**, we can infer the completeness and consistency of information related to the documenting of assigning and reviewing activities (i.e. negative incident).

We performed a proportion test [56] on the four measured values, to identify if teams within scientific software projects perform issue tracking differently from software engineering projects. The test indicates the probability of all proportions being the same. A low p-value indicates there is a significant difference among tested groups. Table 3.8 provides a summary of the test outcome. For **RCR**, it is was more likely for software engineering projects members to exert more effort in documenting the resolvement of issue reports compared to scientific software projects.

Table 3.8: Proportion test of Documentation Quality metric variables

Variables	X-squared	CI, df, p-value	ATLAS-Reco	Belle2	Eclipse	UNICASE
RCR	10365.17	95%, df=3,	0.13	0.00	0.797	0.0086
RCNI	27956.04	$p-value <$	0.560	0.92	0.99	0.78
RA	18953.29	$2.2e-16$	0.71	0.99	1.00	0.92
RNR	40807.42		0.57	0.70	0.0	0.0

Given the proportion of the measured **RCNI** values, the issue reports within software engineering projects have a higher probability to contain more descriptive attributes compared to scientific software projects. For **RA**, ATLAS-Reco has the lowest probability for it to contain assigned issue reports compared to all other projects, which indicates that ATLAS-Reco has the highest amount of incomplete recordings of issue assigning activities. For **RNR**, scientific software projects are more likely to have inconsistent and incomplete recordings of the issue reviewing activities compared to software engineering projects. In conclusion, scientific software projects showed more negative quality incidents and relatively less positive incidents in comparison to software engineering projects.

3.2.1.2 *Interconnectivity*

To determine the differences on how activities interconnect with each other among the investigated projects, we calculated pearson product-moment correlation coefficient of the variable: PMAC belonging to each issue tracking activity described in Section 3.1.3.1. For scientific software projects, members engaged in reporting activity had a weak to no correlation with issue resolvement and management activities. On the other hand, reporters in software engineering projects had a moderate to very strong positive correlation with issue reviewing, prioritizing and resolvement activities. Within all four projects the members involved in resolving activity had a moderate to a very strong positive correlation with issue reviewing and assigning activities. In conclusion,

scientific software projects seem to have a clear separation between roles who report issues and those who resolve and manage them. In other words, people who report issues do not engage in any management or resolvement activities.

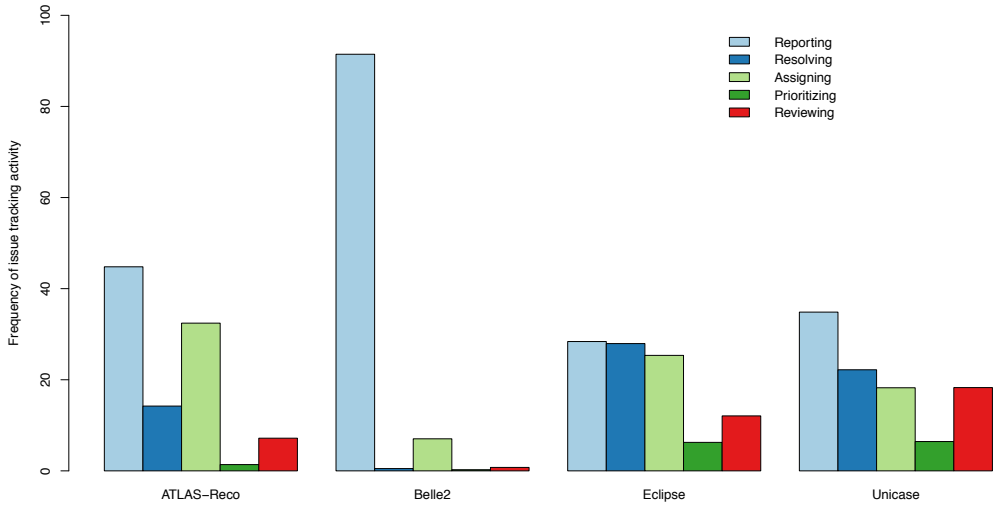

Figure 3.3: Frequency of issue tracking activities (Analyzed)

3.2.1.3 *Frequency*

Figure 3.3 shows the frequency of all the issue tracking activities. The activity frequency is derived from the PMAC variable and the total number of issue tracking activities described in Section 3.1.3.1. The figure indicates that the frequencies within scientific software projects are skewed compared to software engineering projects. The great difference indicated between the frequencies of reporting and resolving activity strongly implies the irregularity and inconsistency of the adopted issue tracking lifecycle within scientific software projects.

3.2.2 *Survey-Results*

This section provides a summary of the analysis outcome from the 16 survey questions represented in Section 3.1.3.2 belonging to 612 participants represented in Section 3.1.4.2. The questions were analyzed to identify the five study metrics: interconnectivity, tools, documentation quality, frequency, enhancements, and problems, which provides a deeper insight on the level of commitment, effort, common procedures used, and behaviors of project members involved in tracking issues.

3.2.2.1 *Demographic*

612 valid responses were collected from August 2011 to August 2012. Figure 3.4 indicates that all of the Belle2 participants are of a physics background as well as the majority of the participants of ATLAS-Reco project (83%). Software engineering is the

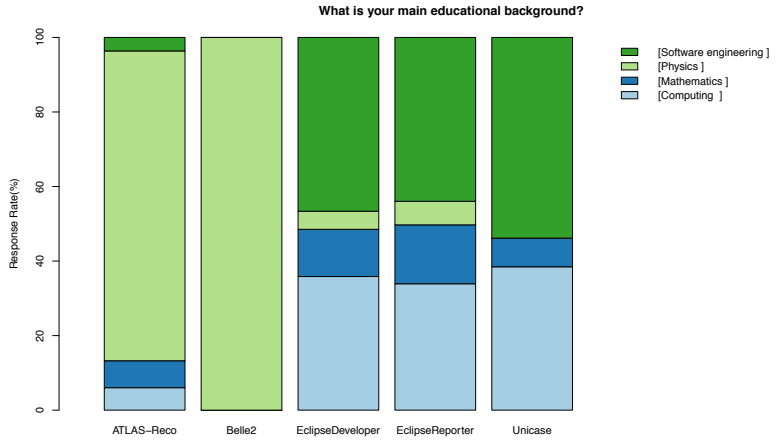

Figure 3.4: Participants' educational background

main educational background for the Eclipse developers, Eclipse reporters and Unicase (47%, 44% and 54% respectively). The next highest background for these three groups is computing (38%,34% and 38.5% respectively). Software engineering presents a small percentage in the ATLAS-Reco (3.5%) and no cases in Belle2. Mathematics background showed the lowest overall percentage, with zero % in Belle2, followed by ATLAS-Reo (7.2%), Unicase (7.6%) , Eclipse developers (12.5%) followed by Eclipse reporters (16%).

Figure 3.5 shows that the highest overall percentage of responses among all projects belonged to the role developers who are also users of the software (51.39% ATLAS-Reco, 46.67 % Belle2, 29.45% Eclipse developers, 15.83 % Eclipse Reporters, 30% UNICASE).

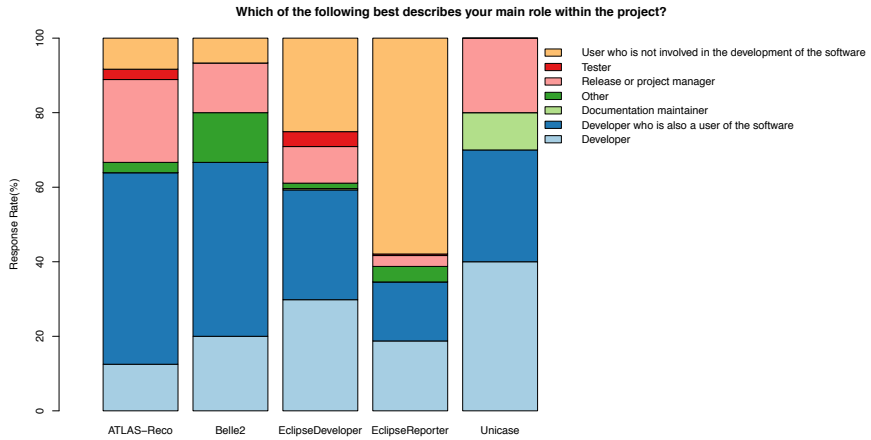

Figure 3.5: Participants' project roles

For the Eclipse reporters, the majority of its responses (58%) belonged to the role of a user who is not involved in the development of the software. Additionally, 22.22% of ATLAS-Reco, 13.33% of Belle2, 9.81% of Eclipse developers, 2.92% of Eclipse reporters

and 20% of UNICASE responses belonged to release or project managers roles. Tester role belonged only to 2.78% of ATLAS-Reco, 4% of Eclipse developer and 0.42% of Eclipse reporters responses. Documentation maintainer role belonged only to 0.4% Eclipse developers and 10% UNICASE responses.

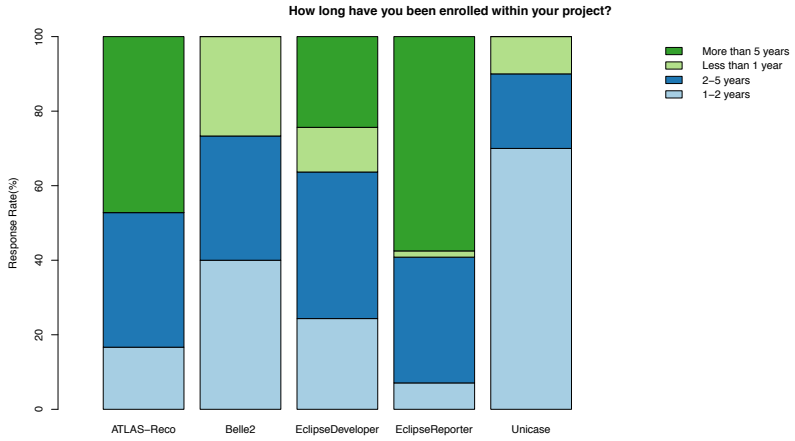

Figure 3.6: Participants' project experience

Figure 3.6 illustrates how long the participants have been enrolled within their projects, 36.1% of ATLAS-Rec, 33.33% of Belle2, 39.32% of Eclipse developers, 33.75% of Eclipse reporters and 20% of UNICASE responses stated they have been enrolled within a period of 2-5 years. While 16.67% of ATLAS-Reco, 40% of Belle2, 24.34% of Eclipse developers, 7.08% of Eclipse reporters and 70% of UNICASE participants have been enrolled within a period of 1-2 years. More than 5 years period of enrollment was reported by 47.2% of ATLAS-Reco, 24.35% of Eclipse developers and 57.5% of Eclipse reporters participants.

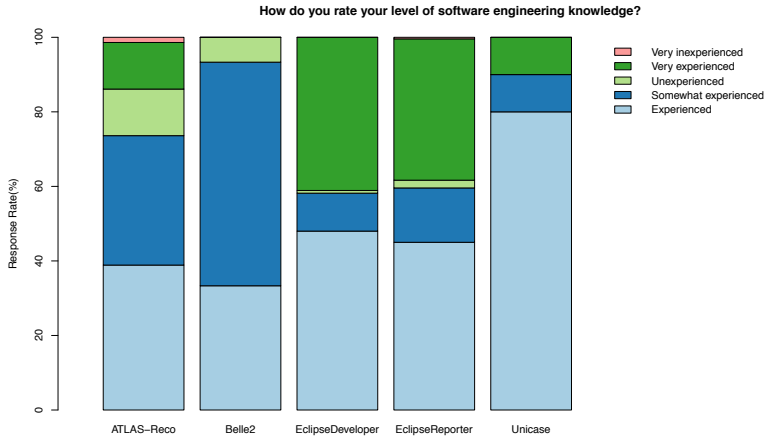

Figure 3.7: Participants' software engineering experience

Figure 3.7 indicates the percentage of participants that were experienced in software engineering within all the projects represented 38.89% of ATLAS-Reco, 33.33%

of Belle2, 48% of Eclipse developers, 45% of Eclipse reporters and 80% of UNICASE responses.While, 34.72% of ATLAS-Reco, 60% of Belle2, 10.18% of Eclipse developers, 14.58% of Eclipse reporters and 10% of UNICASE responses stated that they are somewhat experienced in software engineering techniques. Very experienced participants represented 12%, 10%, 41% and 38% of ATLAS-Reco, UNICASE, Eclipse developer and Eclipse reporters responses respectively. Unexperienced participants represented 12.5% of ATLAS-Reco responses. Very inexperienced participants are the least among all the responses representing only 1.39% of ATLAS-Reco and 0.42% of Eclipse reporters responses.

3.2.2.2 *Interconnectivity*

The participants were asked to specify the issue tracking activities they usually perform on daily basis. Using these responses, we determined the level of interconnectivity between these activities. We calculated the phi coefficient of a binary variable that represents the response of each participant, the variable was set to zero when the participant did not perform a specific activity, and was set to one otherwise. For interpreting the level of interconnectivity we used the scale provided by Salkin [68]. The results indicate that members of scientific software projects involved in prioritizing issues had a moderately positive correlation with resolvement activities and a moderately negative correlation with activities related to discussing issues among team members. ATLAS-Reco and UNICASE members engaged in the documenting of issues tracking activities had a moderately to strong negative correlation with activities related to discussing issues with team members. Members of software engineering projects involved in reviewing activities have a moderately positive correlation between activities related to either reporting or documenting issue tracking activities. While, in scientific software projects members engaged in reviewing activities had weak to moderately negative correlation with resolvement activities. This indicates that scientific software projects have a clear separation between management roles and resolvement and communication activities compared to software engineering projects. For all the projects it was observed that members responsible for resolving or reporting of issues had a moderately positive correlation to the documenting of issues tracking activities. Except for UNICASE, all members engaged in reporting tend to get involved in prioritizing activities. This may indicate that the work flow of how issues are prioritized is different for the other projects with the possibility that managers or resolvers gets to set the issues' priority.

3.2.2.3 *Frequency*

Using the participants responses about the issue tracking activities they usually perform, we calculated the number of times an issue activity was mentioned by participants. After that, we calculate the frequency according to the previously mentioned equation in Section 3.1.3.1. Figure 3.8 represents the frequency of each issue tracking activity in relative to all other activities performed by participants within each project. In scientific software projects, issue prioritizing and scheduling activities tend to have the lowest frequency compared to software engineering projects, as seen within Belle2 project non of participants were engaged in these activities. On the other hand, among

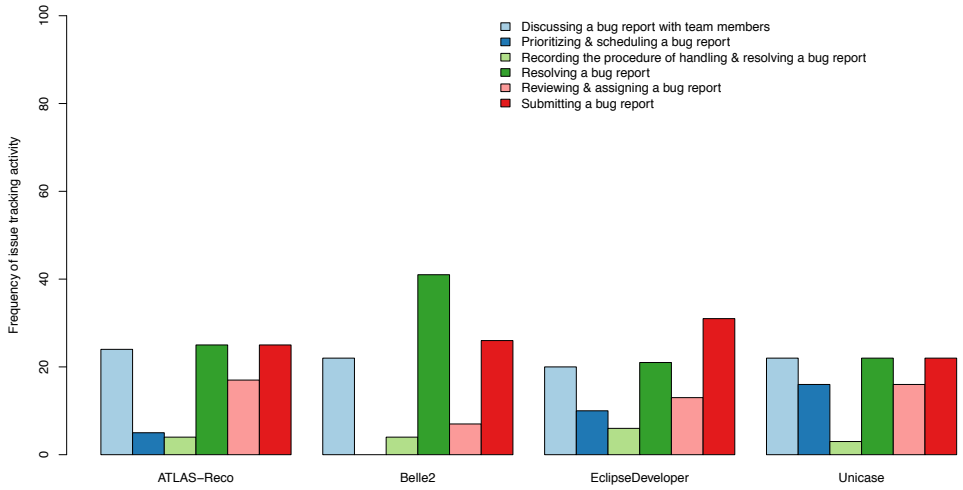

Figure 3.8: Stated frequency of the issue tracking activities (survey)

all the projects the activities related to the recording of issue handling and resolvement had the lowest frequency compared to all other activities. As indicated in Figure 3.8 within all the projects, the ratio between the resolving and reviewing frequencies were within a fairly close proportion (1.3-1.6) in comparison to each other, except for Belle2 project the ratio between resolving (41%) and reviewing (7%) frequencies was larger (5.86) than all the rest.

3.2.2.4 *Quality Ratings of Issue Tracking Practices*

Figure 3.9 illustrates the quality ratings of issue tracking practice within the investigated projects. We have defined four adjective pairs to describe the issue tracking practices in general using a semantic differential scale ranging from (-1 till 2). The par-

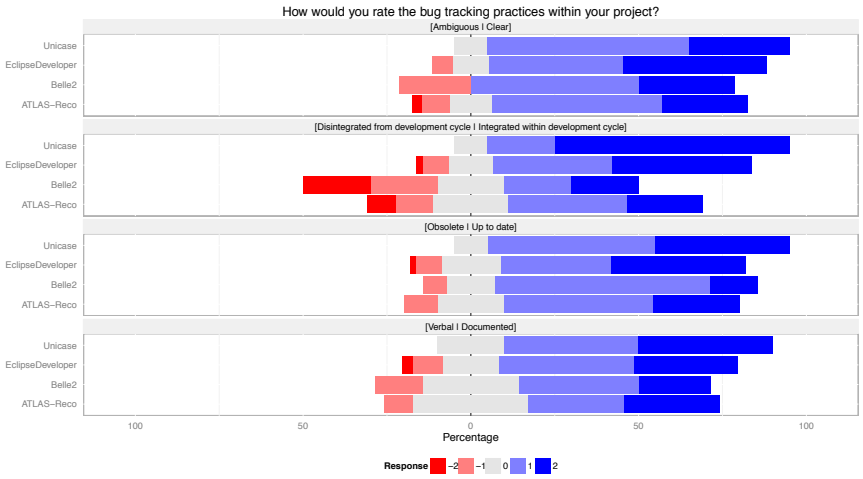

Figure 3.9: Quality ratings of issue tracking practices

ticipants were asked to give an objective feedback on which adjectives best describe the issue tracking practices adopted within their projects.For software engineering projects (83-90%) of the participants stated that issue tracking practices were clearly defined within their projects. While in scientific software projects, (76-79%) stated the same. A higher number of participants (77-90%) within software engineering projects believed that the issue tracking practices applied within projects are integrated within their development cycle in comparison to responses (40-58%) form scientific software projects. More than 70% participants from all projects stated that applied issue tracking practices are up to date. In scientific software projects, 57% of participants believed that their adopted issue tracking practices are documented, while more than 70% participants from the software engineering projects stated the same.

3.2.2.5 *Tools*

Figure 3.10 presents the participants different perspectives on the gained benefit of using a bug tracking system to track and handle software issues. Most of the participants of ATLAS-Reco (40%) and Eclipse (30% & 32%) stated that they benefited from having a unified platform for sharing information regarding software issues.

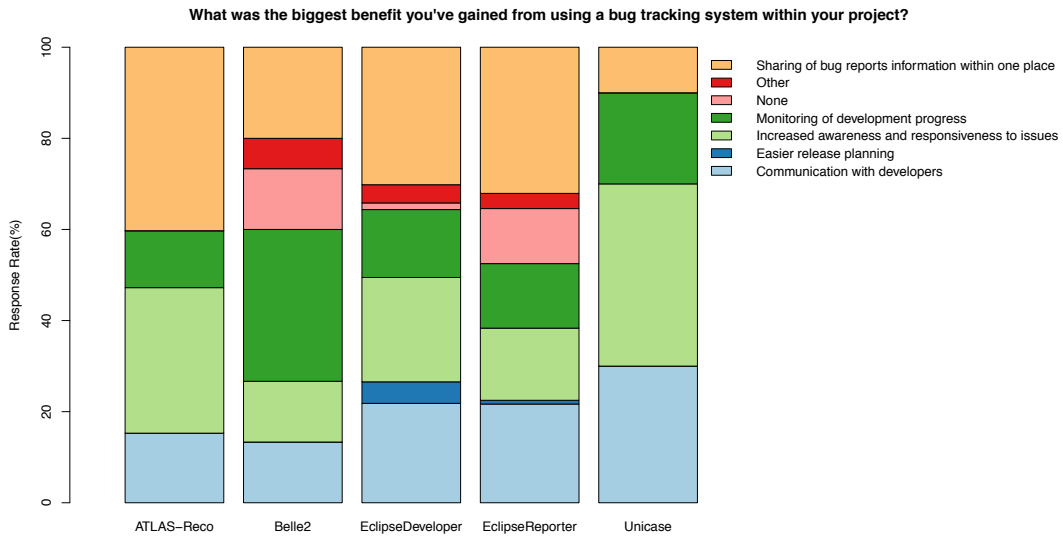

Figure 3.10: Benefit gained from using a bug tracking system (survey)

In UNICASE the largest number of the participants (40%) as well as 22% of Eclipse developers, 15% of Eclipse reporters, and 31% of ATLAS-Reco participants, expressed that the use of a bug tracking system has increased awareness and responsiveness to software issues within their projects. Most of Belle2 responses (33%) as well as 20% of UNICASE expressed that they benefited from monitoring the software development progress. However, 13% of Belle2 and 12% responses stated that they didn't gain any benefits from using a bug tracking system within their projects.

The participants were asked to name the tool they preferred to use in performing each of the issue tracking activities. As shown in Figure 3.11, the results indicate that among all the projects for most of the issue tracking activities bug tracking system was

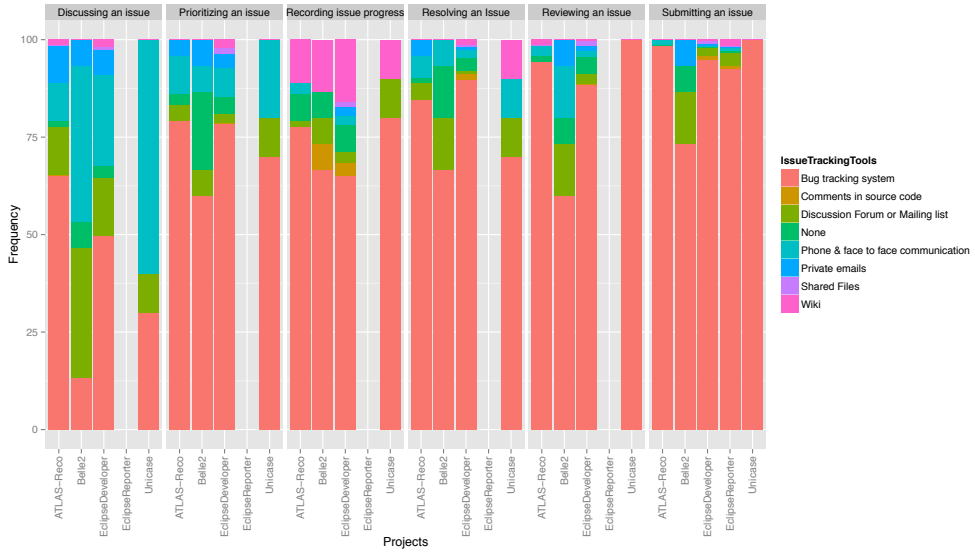

Figure 3.11: Tool preference Vs. performed issue tracking activity

the (60%-100%) participants most favorite tool. However, for discussing an issue report among team members participants in Belle2 (40%), Eclipse (23%) and UNICASE (60%) preferred to perform it using phone and face to face communication. Additionally 33% of Belle2, 15% of Eclipse and 13% of ATLAS-Reco participants preferred using discussion forums or Mailing lists. In Belle2, 13% of the participants preferred reviewing and assigning issue reports either using phone and face to face communication or discussion forums or Mailing lists.

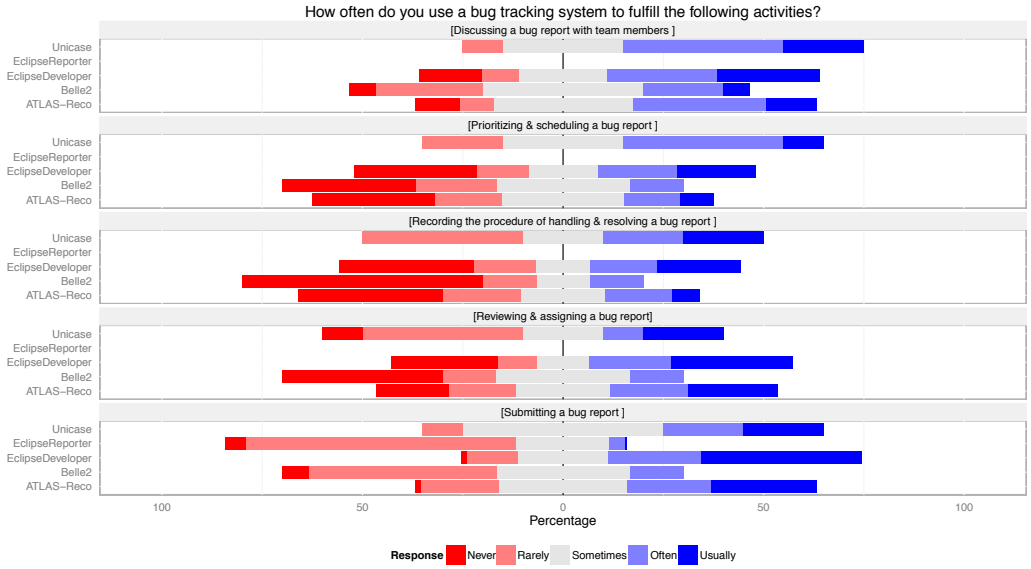

Figure 3.12: Activities performed using a bug tracking system (survey)

Figure 3.12 illustrates how frequently a bug tracking system was used in performing a certain issue tracking activity. We used a Chi-square test for measuring how likely the observed distributions among the investigated projects are due to chance.

The results indicate for 60% of UNICASE, 53% of Eclipse, 46% of ATLAS-Reco and 27% of Belle2 responses stated that bug tracking system was used in discussing issue reports among the development team (insignificant difference p-value = 0.072). For prioritizing and scheduling issue reports in UNICASE projects participants, used it more frequently than the other teams (50% verses Eclipse 39% , ATLAS-Reco 22% and Belle2 13%, p-value= 0.025). When recording the procedure of handling and resolving issue reports 40% of UNICASE , 37% of Eclipse , 24% of ATLAS-Reco and 13% of Belle2 participants used a bug tracking system (insignificant difference p-value= 0.136). For reviewing and assigning issue reports, 51% of Eclipse developers, 30% of UNICASE, 42% of ATLAS-Reco and 13% of Belle2 participants used it (insignificant difference p-value= 0.084). Lastly, for submitting software issue reports, Eclipse developers used it more often than Eclipse reporters and all the other teams (63% verses 4% Eclipse reporters, 40% Unicase, 47% ATLAS-Reco and 13% Belle2, p-value= 0.000)

3.2.2.6 *Problems, Overheads and Enhancements*

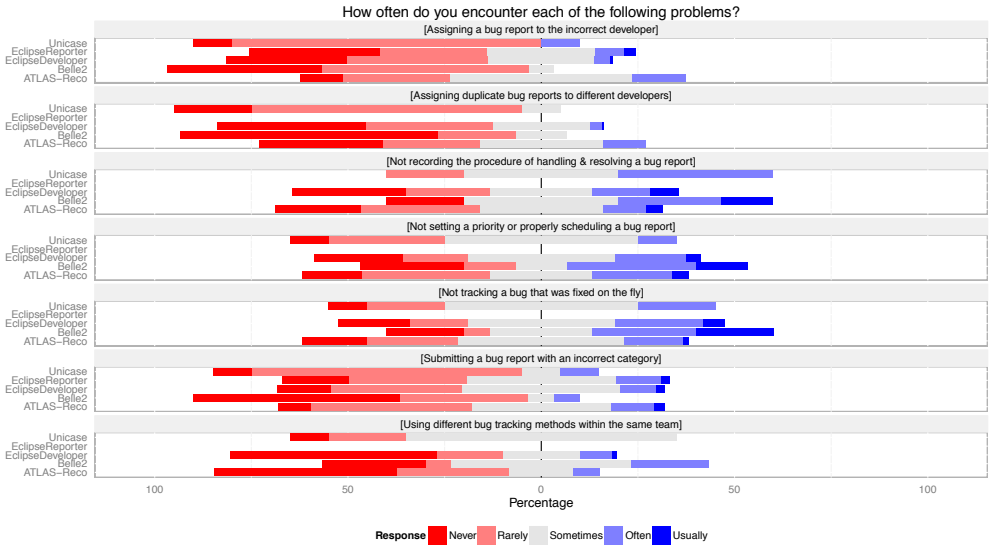

Figure 3.13: Issue tracking problems (survey)

We have defined seven issue tracking problems and asked the participants to state how frequently do they encounter each of them. We used a Chi-square test for measuring how likely the observed distributions among the investigated projects are due to chance. Figure 3.13 indicates how frequent the issue tracking problems were encountered among the investigated projects. The results indicate that assigning a bug report to the incorrect developer was encountered more frequently (usually/often/-sometimes) by ATLAS-Reco participants compared to others (61% of ATLAS-Reco responses verses 38% of Eclipse developers, 32% of Eclipse reporters, 10% of UNICASE and 7% of Belle2 responses, p-value =0.00). Also, assigning duplicate bug reports to different developers was frequently encountered by ATLAS-Reco participants compared to others (43% of ATLAS-Reco responses verses 29% of Eclipse developers, 13% of Belle2 and 10% of UNICASE responses, p-value=0.024). 40% of UNICASE and

Belle2, 22% of Eclipse and 15% of ATLAS-Reco participants expressed that they usually/often encounter an unrecorded procedure of handling and resolving a bug report (insignificant difference p-value = 0.065). Not setting a priority or properly scheduling a bug report was encountered more frequently (usually/often) in Belle2 compared to other projects (47% verses 25% ATLAS-Reco, 22% Eclipse developers and 10% UNICASE responses, p-value=0.040). 60-73% of all participants mentioned that they frequently encounter (usually/often/sometimes) untracked bug reports that were fixed on the fly without being documented. Submitting a bug report with an incorrect category was mostly common to occur in ATLAS-Reco and Eclipse projects compared to others (63% of ATLAS-Reco, 52.5% of Eclipse Reporters and 52% of Eclipse Developers responses verses 20% UNICASE and 13% Belle2, p-value=0.010). Belle2 participants are more likely (usually/often) to use different bug tracking methods within the same team compared to other projects. (20% of Belle2 participants verses 9% of Eclipse developers and 7% of ATLAS-Reco, p-value=0.010).

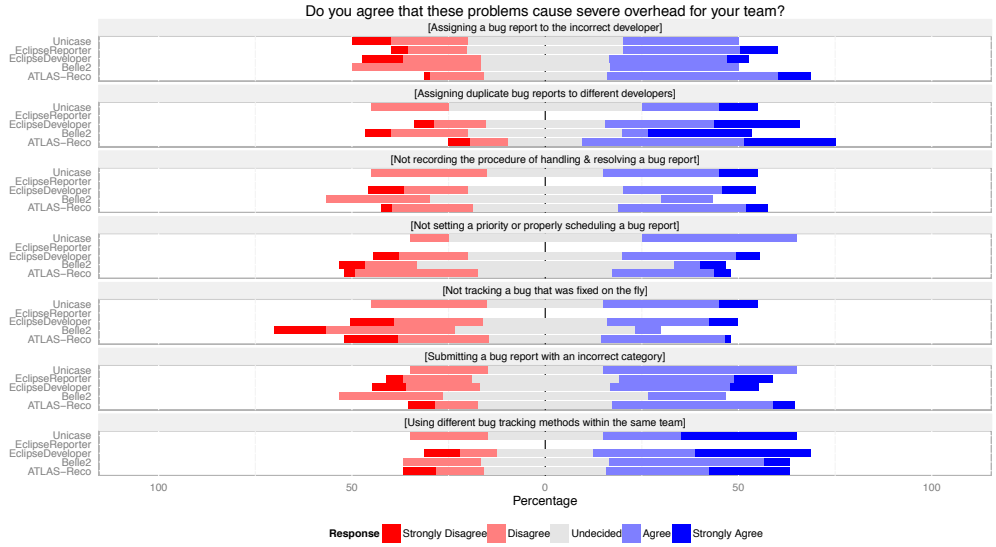

Figure 3.14: The significance of issue tracking problems (survey)

We have asked the participants to state their opinion on whether they agree or not that certain issue tracking problems causes significant overheads on the development team. We used Chi-square test to measure the statistical significance of differences among projects. Figure 3.14 indicates the level of agreement among participants regarding the significance of overheads caused by issue tracking problems. The results indicate that ATLAS-Reco had the highest level of agreement on the significance of overheads caused by assigning a bug report to the incorrect developer than the other projects (53% verses 40% Eclipse Reporters, 36% Eclipse Developers, 33% Belle2 and 30% UNICASE, p-value=0.030). 65% of ATLAS-Reco, 51% of Eclipse developers, 33% of Belle2 and 30% of UNICASE participants agreed on the significance of overhead caused by assigning duplicate bug reports to different developers (difference is insignificant p-value=0.285). 13–40% of all participants agreed on the importance of the overheads caused by not recording the handling and the resolving of issue reports. 40% of UNICASE, 36% of Eclipse Developers, 31% ATLAS-Reco and 13% of Belle2

agreed (strongly-agreed/agreed) that not setting a priority or properly scheduling a bug report imposes a significant overhead on their teams. 40% of UNICASE, 33% of ATLAS-Reco and Eclipse participants agreed that bug reports that were fixed on the fly without being documented imposed a significant overhead on their teams, while only 7% of Belle2 participants agreed to that. 50% of UNICASE, 47% of ATLAS-Reco, 40% of Eclipse reporters and 38% of Eclipse developers participants agreed on the significance of overheads imposed by submitting a bug report with an incorrect category, while only 20% of Belle2 participants agreed to that. 47–56% of all participants agreed on the importance of overheads imposed when using different bug tracking methods within the same team.

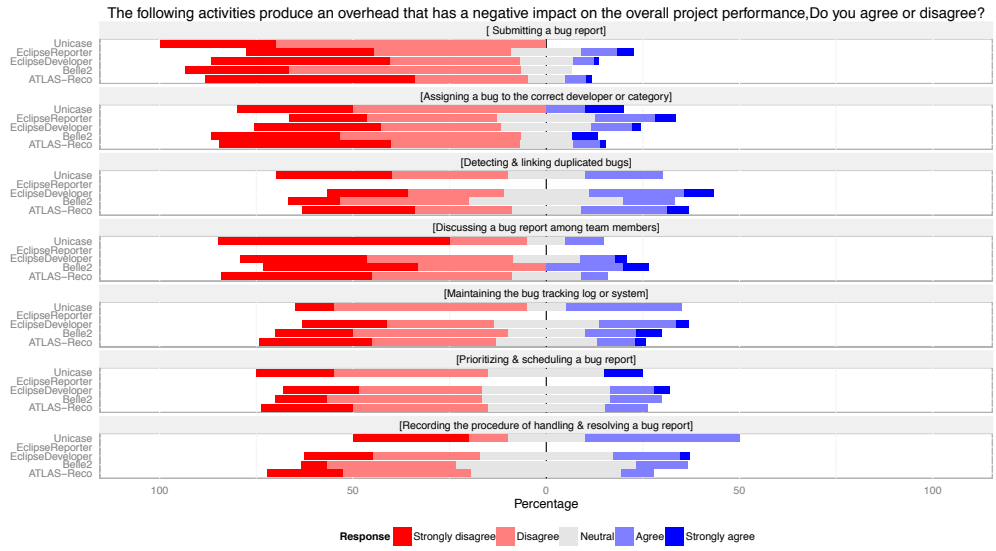

Figure 3.15: Issue tracking activities that are considered as an overhead

Participants were asked to state their opinion on whether they agree or not that certain issue tracking activities produce an overhead that has a negative impacts on the overall project performance. We used Chi-square test to measure the statistical significance of differences among groups. Figure 3.15 illustrates the level of agreement among participants regarding the issue tracking activities that imposes a negative impact on team performance. The results indicate that UNICASE had the highest level of disagreement compared to other groups on submitting a bug report imposing a negative impact on team performance (100% verses 87% Belle2, 83% ATLAS-Reco, 80% Eclipse developers and 68% Eclipse reporters, p-value=0.008). While, UNICASE and Belle2 participants had the highest level of disagreement among other projects on assigning issue reports to the correct developer having negative impacts on project performance (80% verses 78% ATLAS-Reco, 64% Eclipse developers and 54% Eclipse reporters, p-value=0.004). The level of disagreement on discussing a bug report among team members having negative impacts was generally high overall (80%–71%). The level of disagreement on the following activities: (1) maintaining the bug tracking log or system, (2) prioritizing and scheduling a bug report and (3) detecting and linking duplicated bugs, imposing negative impacts was generally the same (45%–60%). 53% ATLAS-Reco, 45% of Eclipse developers, 40% of UNICASE and Belle2 participants

disagreed that recording the procedure of handling and resolving a bug report places reduces teams performances.

Table 3.9: Number of open ended responses regarding issue tracking problems or overheads

Project	# of responses	# of valid responses
Eclipse	127	95
UNICASE	1	0
ATLAS-Reco	9	9
Belle2	1	1

Table 3.9 indicates the total number of participants open ended feedback related to project specific issue tracking problems or overheads that were not mentioned in the survey. We have reviewed each of the submitted responses and came up with specific categories that represent their context. The responses were categorized into 17 issue tracking problems and 18 overheads categories.

Table 3.10: Issue tracking problem categories vs. number of open ended responses

	# of Responses	
Problem Categories	**Eclipse**	**ATLAS-Reco**
Rejecting bug reports without thorough examination	3	0
Receiving negative or no feedback from developers regarding a submitted bug	5	1
Bug Tracking System Poor Usability & Functionality	15	1
Delayed bug resolvement & feedback	9	1
Lack of traceability between duplicated bug reports	3	1
Unresolved bug reports that are obsolete	2	1
Submitting bug reports to inactive team	1	0
Lack of integration between development, project management & collaboration tools	16	4
Lack of traceability between bug reports & its target release	3	2
Lack of transparency in release planning & workload scheduling	2	1
Ineffective planning and administration of bug reports' lifecycle	3	0
Difficulty in decomposing complex bugs into several tasks	2	0
Conflicts between team members regarding the use of project management tools	1	0
Lack of support for release planning	1	0
Losing track of long discussion on bug tracking system	1	0
Unrecorded change within bug reports' status	2	0
Reopening bug reports without investigating its relevance to the arising problem	1	0

Table 3.10 represents the number of participants' open ended feedback related to any of the 17 issue tracking problem categories, which included only responses from Eclipse and ATLAS-Reco projects since the responses from UNICASE and Belle2 projects did not contain any information related to issue tracking problems.

The problem category having the highest percentage of responses among both projects (16.84% in Eclipse and 44.44% in ATLAS-Reco) was the lack of integration

between development, project management and collaboration tools. Most of the participants expressed their discontent of not having interconnectivity among these tools. For example, if a project member would like to assign a bug report to the last person who has caused a change in the affected part of the source code, then he has to look up this information within the version control system and then update the bug report via bug tracking system with corresponding information. In addition, some responses mentioned that having no real integration among bug tracking system, IDE and project planning tools (e.g. agile) increases conflicts and inconsistency of information. In Eclipse, 15.79% of responses indicated that it is sometimes difficult and unpleasant for the participants to use some of the bug tracking system features, i.e. the system had a poor usability and functionality. In ATLAS-Reco, 22.22% of responses indicated that there is no traceability between bug reports and target releases, which made release planning a more difficult and complex process. On the other hand, in both projects (9.47% in Eclipse and 11.11% in ATLAS-Reco) the participants stated that it takes a long time for the bugs to get fixed or it takes the developers a long time to give a feedback regarding an inquiry.

Table 3.11: Issue tracking overheads categories vs. open ended responses percentage

	# of Responses		
Overhead Categories	**Eclipse**	**ATLAS-Reco**	**Belle2**
Prioritizing bug reports	1	0	0
Bug reports notifications	1	0	0
Finding the right category	4	1	0
Finding the right assignee	1	1	0
Finding if a bug is a duplicate or not	7	1	0
Writing reproducible & comprehensible bug reports	2	1	0
Triaging obscure incomplete & invalid reported bugs	18	1	1
Keeping bug reports' status up-to-date	5	1	0
Feature Requests submitted as bugs	7	0	0
Submitting bug reports that can be fixed right away	2	0	0
Bug reports concerning backwards compatibility	1	0	0
Recovering the resolution of untracked bug	1	1	0
User communicating with developers through private emails	1	0	0
Managing patches submitted by non-project contributors	1	0	0
Managing a bug to be fixed in multiple releases	1	1	0
Verification of resolved bugs	2	0	0
Following up on unfinished tasks with developers	0	2	0
Unassigned bug reports	0	1	0

Table 3.11 represents the number of participants' open ended feedback related to any of the 19 issue tracking overhead categories, which included only responses from Eclipse, ATLAS and Belle2 projects as responses from UNICASE project where invalid. A common overhead among all three projects was the process of triaging unclear and invalid bug reports. Participants consider it a very arduous and time consuming task to either figure out the problem itself or identify if it was an invalid claim. Only in the ATLAS project, 22.22 % of the participants mentioned that it is an overhead to follow up on unfinished tasks with developers. Team members stated that they are overloaded with their own research, and they tend to avoid software resolving tasks

that is not relevant to their research work. Only in eclipse project 7.37% participants stated that it is an overhead to deal with feature and change requests that are submitted as bug reports, not having a clear distinction that the report describes a feature request makes it harder for the triager to process it. In both Eclipse and ATLAS project, 7.37% to 11.11% of the participants stated that it is a hard and tedious process to find out if an issue report is a duplicate or not. Only in the ATLAS-Reco project, 11.11% of the participants stated that having unassigned bug reports is an overhead that, can cause issue reports not be resolved or even increase their resolution time. Since it is hard with having no direct trace of the issue report's assignee to communicate with the person in charge.

Table 3.12 illustrates the number of open ended responses regarding the participants suggestions on how to enhance the current issue tracking activities. We have reviewed each of the submitted responses and came up with specific categories that represent their context. The responses were categorized into 9 enhancement categories.

Table 3.12: open ended responses regarding issue tracking enhancements

Project	# of responses	# of valid responses
Eclipse	10	7
UNICASE	0	0
ATLAS-Reco	2	2
Belle2	3	3

Table 3.13 represents the open ended responses belonging to the 9 issue tracking enhancement categories within Eclipse, ATLAS, and Belle2 projects.

Table 3.13: Issue tracking enhancements categories vs. open ended responses percentage

	# of Responses		
Enhancement Categories	**Eclipse**	**ATLAS Reco**	**Belle2**
Automatically detect & link duplicated issue reports together	2	1	0
Automatically assign a suitable assignee or category to a newly reported bug	1	1	0
Linking the developers version control activities to the bug tracking system	1	0	0
Integrating development, project management & collaboration tools together	2	0	0
Integrating the bug reporting system with the developed software product	2	0	0
Ensuring a well documented process of issue reporting & resolving	1	0	0
Enhance or replace the bug tracking system to achieve a better usability & functionality	3	2	1
Document minor development bugs as a comment in source code.	0	0	1
Issue assigning and reviewing to be performed by a specialized & dedicated developer	0	0	1

In all three projects (42.86% in Eclipse, 100% in ATLAS and 33.33% in Belle2), most of the participants stated that the bug tracking system usability and functionality

need to be enhanced in order to achieve a more effective and efficient issue tracking practice. In ATLAS (50%) and Eclipse (14.29%-28.57%), of the participants suggested that some parts of the reviewing and assigning of issue reports should be automated, in particular: (1) the detection and linking of duplicated issue reports, and (2) the determination of a suitable assignee or a category to a newly submitted issue report, as they consider them tedious and time consuming activities.

In the Eclipse project, 28.57% of the participants suggested that the IDE should support the ability to directly report software issues instead of using Bugzilla's web interface to make issue reporting a less time consuming activity.

In addition, they suggested the tight integration between the development and project management tools to reduce conflicts between the different activities. For example, if a project member was planning a meeting on reviewing a software release using a team collaboration tool, through this tool he should be able to add information about software issues that are stored in issue tracking repositories. In Belle2 33.33%, of the participants suggested that issue assigning and reviewing should be developer-based activities, in which a specific developer is exclusively dedicated to their performance.

3.3 SUMMARY

This section summarizes the significance of our findings along with an interpretation of the found similarities and differences when comparing issue tracking practices in scientific and software engineering domains.

3.3.1 *Main Similarities*

Regardless of the level of interconnectivity between issue tracking activities, participants of both project domains who were involved in issue resolvement tended to be involved in management activities as well. This shows that issue resolvement is an influential activity that involves individuals who possess enough project organization and domain knowledge that enables them to handle and manage software issues.

Large scale projects from different domains tend to have commonalities in tool usage, as well as issue tracking problems and encountered overheads. That case was strong for ATLAS-Reco and Eclipse. As seen in Section 3.2.2.6 and Section 3.2.2.5, most of ATLAS-Reco and Eclipse participants stated that they benefited from having a unified platform for sharing information regarding software issues. Additionally, submitting a bug report with an incorrect category was more common in ATLAS-Reco and Eclipse projects than to all other projects. Regardless of the domain differences the two projects share many similarities in terms of project organization and adopted issue tracking practices. We can assume that these similarities are due to the fact that both projects share a globally distributed software development paradigm.

3.3.2 *Main Differences*

There was a large difference in the level of interconnectivity of issue tracking activities between both project domains. Participants from software engineering projects tended to be engaged in multiple activities, whereas participants from scientific projects fo-

cused on single activities. This indicates a higher level of collaboration within software engineering projects in comparison to scientific software.

The value of issue tracking activities frequency obtained from the survey and the analyzed issue tracking repositories indicated an irregular distribution within the studied scientific projects. Some activities had frequencies that are inconsistent with the normal distribution of a regular issue tracking activity lifecycle. For example, in Belle2 reporting formed the higher frequency in comparison to resolving and managing. This demonstrates that a lot of issue reports were unprocessed and just closed. This is an indication that the documentation of issue tracking activities is more balanced in the software engineering projects and that it tends to be inconsistent within the studied scientific projects.

Scientific software projects showed more negative documentation quality incidents and relatively less positive quality incidents in comparison to software engineering projects. The members of software engineering projects exerted more effort in documenting the resolvement, as well as in the reporting and reviewing activities. The documentation within software engineering projects contained more consistent, descriptive, and useful information that can indicate how the issue progressed while performing these issue tracking activities. Additionally, ATLAS-Reco had the highest amount of incomplete recordings of issue assigning activities compared to other projects. Furthermore, the surveyed participants of ATLAS-Reco stated that one of the major projects' overheads was due to assigning an issue report to an incorrect developer. This confirms the findings of Shokripour et al. [75] that the time spent in examining each submitted bug report and deciding about how the report will be organized within the development process of the software project represents an overhead to the project.

There was a noticeable difference between both domains in the quality ratings of issue tracking practices, in which less number of participants in scientific software projects believed that their issue tracking practices were documented in comparison to participants from software engineering projects. This indicates that scientific software participants are actually aware of the shortage they have in project management artifacts. The result was also confirmed by our data analysis results and corroborates the findings of the study of Pawlik et al. [61].

Our results, showed an interesting correlation with the observation stated by Pawlik et al. [61] that documentation related to planning and project management is produced more for larger scientific software projects in comparison to other project scales. For the case of ATLAS-Reco (large-scale) and Belle2 (small-scale) projects, the frequency of issue tracking activities within ATLAS-Reco seemed to be more balanced than Belle2, where there was a great difference between the frequencies of reporting and resolving activity. This implies a lot of irregularities and inconsistencies of the adopted issue tracking documentation activities within Belle2, where less activities are actually being documented in comparison to ATLAS-Reco.

Regarding the documentation quality metrics measured for Belle2 and ATLAS-Reco especially for positive and negative incidents. Belle2 showed a lower rate of issue reports with commented resolvement *(RCR)*, in comparison to ATLAS-Reco. This indicates that Belle2 members exert a lower amount of effort in documentation when resolving issues in comparison to ATLAS-Reco. Additionally, Belle2 showed a higher

rate of closed issue reports having no resolution (RNR), indicating a higher rate of information inconsistency. However, Belle2 showed more positive documentation quality incidents regarding the rate of issue reports that contain descriptive attributes (RCNI) and rate of assigned issue reports (RA) in comparison to ATLAS-Reco. This indicates the existence of more descriptive and informative issue reports within Belle2.

Our study results indicate that scientific software projects have problems regarding the documentation of issue tracking activities, participants from the studied software engineering projects had ineffective planning and administration in the bug reports' lifecycle, difficulty in decomposing complex bugs into several tasks and incidents of reopening bug reports without investigating its relevance to the arising problem. This indicates a shortcoming in the planning area of issue tracking within software engineering projects.

In summary, scientific projects tend to have more problems in the documentation and triaging processes, whereas software engineering projects tend to have problems on the management and planning levels. The differences in the frequency of issue tracking documentation and planning activities, indicate that issue tracking tools might need to be tailored to the specific domain needs of the individual projects. For example, for software engineering projects issue tracking tools might need to be enhanced with scheduling and management features. Issue tracking tools for scientific projects should provide additional features for administering the existing documentation, and aid the participants in decisions related to assigning and validating the uniqueness of an issue (i.e., identify if the issue was a duplicate or not), to encourage them to exert their effort in issue tracking.

3.3.3 *Specific Domain Issue Tracking Practicses*

We developed *INExPERT*, a technique for assignee recommendation which is described in Chapter 4, based on the following two aspects. First, the fact that any reduction in the time spent on assigning issue reports frees resources for software product improvement [75]. Second, the results of our study showed that the type and frequency of activities performed by project members can indicate their expertise and the level of involvement within the project. *INExPERT* leveraged these aspects by inducing the roles and expertise of suitable assignees. Additionally, *INExPERT* aims at encouraging the collaboration among project members of scientific software projects by making it easier for software issues to be allocated to a specific project member, while at same time hiding the technical and managerial complexities of the assignment activity.

4

ISSUE ASSIGNEES RECOMMENDATION TECHNIQUE-INExPERT

Issue-tracking repositories are widely used Software Configuration Management (SCM) tools. Apart from serving their primary purpose of storing issue reports, they are occasionally used as a database of feature requests and simple TO-DOs. Especially in open-source projects, these repositories are accessible to the end-users; allowing them to collaborate directly with the developers. This possibility of collaboration from various stakeholders helps identify relevant features and improves the quality by allowing more issues to be identified [65]. However, this advantage comes with significant costs [2], because every new issue report has to be triaged. An important task in the triaging process is the assignment of an issue report to a developer.

A number of approaches exist to semi-automatically identify and recommend developers, e.g. using machine learning techniques, social network-based approaches, and traceability between software development artifacts [2][31][87]. These approaches mine history data of issue-tracking repositories to create a ranked list of recommended assignees. Assignees in the list are either good candidate(s) for working on the issue report themselves, or they have the expertise to assign it to a suitable developer. The list does not include any unqualified/non-matching candidates, i.e. "tossing" effect [38].

In this work, we describe a new approach for assignee recommendation named *INExPERT*, that leverages user activities in an issue-tracking repository. Mining history of all the activities (i.e. review, assign, and resolve) done by a user within the issue-tracking repository can indicate his role, expertise, and involvement in the project to some extent. For example, if the user's activity logs indicate that most of what he does is assigning issue reports, this represents that he is either a project manager or an issue triager. Activities done in an issue-tracking repository are divided into three main categories: reporting, managing, and resolving issue reports. Reporting an issue report includes capturing and communicating identified problems as issue reports. Managing an issue report is the activity of deciding on its progress. Finally, resolving an issue report is concerned with the implementation of its resolution. These activities are derived from a set of heuristics obtained from previous studies [3][76] [82][55][25].

INExPERT constructs an activity profile for each issue-tracking repository user. Candidate assignees are identified based on the relevance of their expertise in the topic of the new issue report, and are then ranked using the activity profile. For example, a project member associated with a certain type of issue (e.g. memory leak related issues) can be included in the list of assignee candidates for a new issue in the same category, and is then ranked higher if his activity profile is that of a manager specialized in assigning similar issue reports. With an activity profile of a manager, he will either have enough expertise to work on the issue himself; or reassign it to a more suitable developer. So, by having a list of recommended assignees, less experienced (or new) project members can still be able to triage issue reports. Further, by saving time and effort, it can benefit volunteer-based triaging process.

To evaluate the applicability of *INExPERT*, the research team has applied it to issue reports from three different projects: (1) a software engineering research project, (2) an open-source project with market orientation, and (3) a scientific software research oriented project. The first two projects have a developer-based triaging process, where a specific developer is obliged to administer issue reports. The third project relies on a volunteer-based triaging process, where the responsibility of assigning issue reports is stipulated among developers. We compared our results to a state-of-the-art approach [77] that combines a supervised learning model based on Support Vector Machines (SVM) [27] and an unsupervised generative model based on LDA [79]. Results of the comparison have shown that *INExPERT* outperforms the LDA-SVM–based approach. Additionally, in contrast with LDA-SVM, *INExPERT* is able to provide sensible recommendations for any issue report having a resolver who was not present in the training set.
Chapter content:

- An assignee recommendation approach, *INExPERT*, which utilizes the activity profiles of the users of an issue-tracking repository to identify roles, expertise, and involvement in the project. The obtained heuristics are used for ranking developers in the candidate list.
- An insight into the pattern of activities performed by developers in issue-tracking repositories of various projects. These patterns detail how various users interact with issue reports and how the triaging process is done in a certain project.
- A formative evaluation of *INExPERT* that incorporated a Quasi experiment and structured interview.
- Domain experts' feedback on the accuracy of *INExPERT*.
- A comparison between the results obtained from *INExPERT* and the LDA-SVM–based approach.

The Chapter is organized as follows: Section 4.1 presents our approach of assignee recommendations: *INExPERT*. We detail the steps of: (1) categorizing issue reports into topics, (2) creating an activity profile for users, (3) assignee recommendation(s) for a new issue report, and (4) assignee ranking. Section **??** presents a case study, which investigates the use of *INExPERT* within a scientific software project. The outcome of this investigation consists of qualitative and quantitative data that was collected by performing a *pre-post testing quasi experiment* and a *structured interview*. Section 4.3 gives an overview of the evaluation setup and the dataset. Subsequently, it presents a summative evaluation of *INExPERT* along with its comparison with the LDA-SVM–based approach. Additionally we obtained a direct feedback from three scientific domain experts regarding the accuracy of *INExPERT*.

4.1 INEXPERT DESIGN

In order to provide a list of recommended assignees, we start by categorizing all the issue reports from our dataset. We achieve this using a topic model that groups the issue reports into topics (see Section 4.1.1). Next, we mine the issue-tracking activities

for each project member (activity profile) to determine their topic associations (see Section 4.1.2). When a new issue report is submitted, and is in the process of assignee recommendation (see Section 4.1.3); we first determine the topics this issue report is associated with (see subsection 4.1.3.1), and then generate a list of recommended assignees that match the new issue's topics (see subsection 4.1.3.2). Finally, we rank the developers based on the type and frequency of the issue-tracking activities they perform (within topics similar to the newly reported issue's) (see Section 4.1.4).

4.1.1 *Categorizing Issue Reports into Topics*

In this step, LDA [79] (an unsupervised generative model) is used to cluster issue reports into topics. We create a *Topic Model* for each issue report. This is done by categorizing the terms (i.e. words within issue reports) into clusters (topics). We only consider *title*, *description*, and *system component* of the issue report; as these features contain the most relevant detail of the problem (namely, release and component information). Selected features of the issue reports are tokenized using Apache Lucene tokenizer[1]. Subsequently, the tokenized data is filtered by removing stop words, HTML tags, hexadecimal numbers, and numerical information. The filtering process is done in an iterative manner. Regular expressions and term elimination methods are created and adjusted for each iteration to make sure that terms that belongs to parts of speech such as nouns, verbs, adjectives and adverbs are not eliminated. The terms are additionally processed by a lemmatizer from Stanford CoreNLP tools[2] for grouping the different inflected forms of a word. For example, the term "better" and "good" are grouped into a single term "good". Consequently, this pre-processing reduces the noise in the dataset, and increases the quality of topic categorization.

Next, we use a topic modeling MATLAB toolbox[3] to create the topic model. A constraint is added so that each issue report should only be associated with a fixed number of topics, and each topic should be associated with a fixed number of terms. However, a term can be associated with more than one topic. The LDA model assigns terms to topics based on the Gibbs Sampler approach [26]. This approach first assigns any topic randomly to a term. Then, iteratively, it assigns a topic to each term based on (1) the number of times the term is related to the topic in the issue report, and (2) the number of times the issue report is related to the topic. The final output of the LDA computation is $R \times T$ matrix namely $B_{R\times T}$, where R represents the total number of issue reports, and T represents the total number of topics. Each column in a given row $B_{i,j}$ contains the number of times a certain topic t_j is associated with a certain issue report r_i.

From $B_{R\times T}$, given the value of each row B_i, it would be difficult to induce significant information, such as knowing if a certain issue report r_i is more relevant to a topic t_j in comparison to other issue reports. To be able to draw comparisons among topics associations in issue reports from $B_{R\times T}$, the value belonging to any given row and column $B_{i,j}$ is normalized to assign a fraction of its value (i.e., topic association of r_i with t_j) against the rest of the values in B_i.

1 http://lucene.apache.org/
2 http://nlp.stanford.edu/software/corenlp.shtml
3 http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm

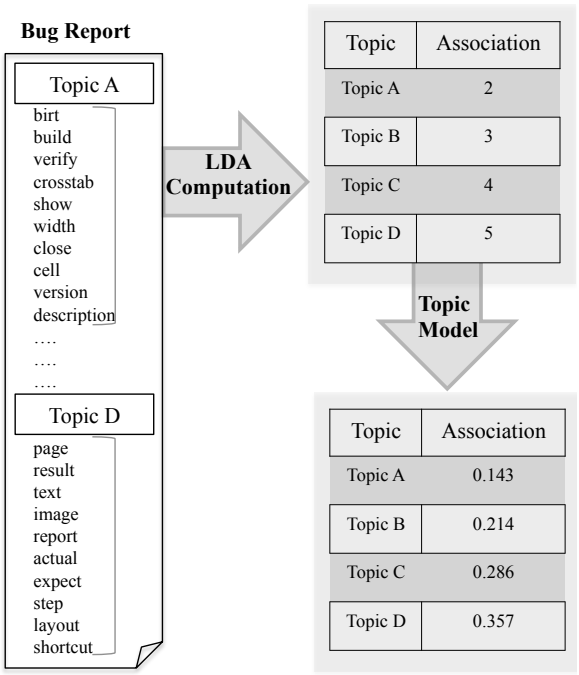

Figure 4.1: Issue Report's Topic Model

Figure 4.1 shows an example of a given 4-topic model that has topics A, B, C & D extracted using LDA. Each topic is associated with only 10 terms. Where a given issue report r_i having the topics associations within B_i : 2, 3, 4 & 5. From this output we can deduce that topic A is related to issue report r_i twice, topic B is related to issue report r_i thrice, and so on. As a result, 14 (2+3+4+5) represents the total number of times all four topics are related to issue report r_i. Therefore, the ratios are 2/14 for $B_{i,A}$, 3/14 for $B_{i,B}$, 4/14 for $B_{i,C}$ and 5/14 for $B_{i,D}$.

4.1.2 *Issue-Tracking Activity Profile*

After categorizing issue reports into topics, we create an *Activity Profile* for each user of the issue-tracking repository, by mining history logs and issue report topic models (as shown in Figure 4.2).

A user's activity profile consists of two parts: (1) *User's roles* (see subsection 4.1.2.1), which indicates the activities done by him (e.g. review, assign, and resolve), and (2) *User's topic associations* (see subsection 4.1.2.2), which indicates his involvement with the issue reports. We detail each of these parts in the next sections.

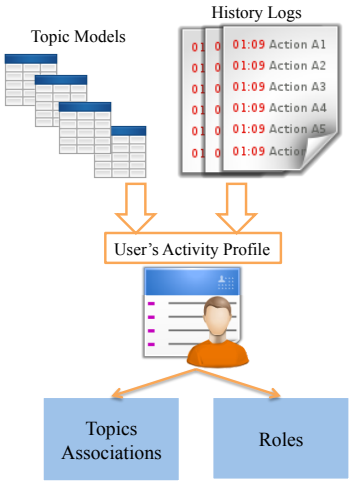

Figure 4.2: User's Activity Profile

4.1.2.1 *User Roles*

In an issue-tracking repository, users can perform various activities, as per their role in the project. Figure 4.3 presents a typical activity-role scenario. While this scenario might be valid in a project-specific context, users often play multiple roles in the same project. Hence, the activities they perform in the issue-tracking repository are not a precise representative of one single role.

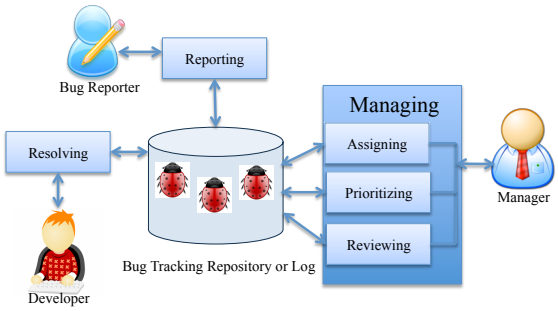

Figure 4.3: Issue-Tracking Activities

For example, a developer can assign, as well as resolve issue reports; and a manager can also be a developer. In this work we consider three activities: reviewing, assigning, and resolving; relevant for identifying assignees who either have enough knowledge to resolve the issue, or to reassign it. We leave out reporting as it does not add value to the expertise determination (actually, even end-users or clients are allowed to report new issues in various projects). Prioritizing an activity is left out as it is often done as part of the reviewing process. For each of the three activities we consider as relevant, a user can have the role of an assigner, a reviewer, or a resolver.

Within a typical software project, an issue report goes through a standard lifecycle; in which it gets submitted, assigned, resolved, verified, and finally closed. The issue-tracking activities are actions done to transfer an issue report from one of these states into another. However, the way each of these activities is represented within different issue-tracking repositories may vary, due to the fact that issue reports' attributes may

differ from an issue-tracking repository to another. For example, in a certain issue-tracking repository, two attributes (namely, status and resolution) are used to indicate a trace to the issue report's status (e.g., an issue can have a "Resolved" *status* together with a "Fixed" *resolution*, while in another issue-tracking repository the *status* attribute is the only indication). The main question to be answered next is how to identify an issue-tracking activity. We identify an activity as a specific pattern in the history logs of the issue-tracking repository. For example, an assigning activity is *acknowledged* if the history log indicates that the issue report's attribute referring to the *assignee* (the person involved in resolving the issue) has been updated with a valid assignee ID. Therefore, the pattern that indicates the occurrence of an activity depends on the issue reports' attributes within the issue-tracking repository. However, we have formulated a set of heuristics rules that define the patterns of history log entries that can be further adjusted to fit the issue reports' attributes of a certain issue-tracking repository. The heuristics rules used to identify the occurrence of these activities are summarized in Table 4.1.

Table 4.1: Heuristics Rules Used for Identifying Issue-Tracking Activities

Activity	Rule
Assigning	*If* the history log records a change in the issue report's attribute referring to the person involved in resolving the issue (i.e., assignee) *Then* the user responsible for that change has performed an assigning activity (i.e., has assigned/reassigned an issue report to an existing assignee).
Resolving	For an *already assigned* issue report, *if* the history log records a change made by the issue report's assignee (that change should be in the issue report's attribute referring to its resolution status, and should indicate that a formal resolution has been reached (e.g. fixed/won't fix/duplicate)) *Then* the user responsible for that change has performed a resolving activity.
Reviewing	For an *unassigned* issue report, *if* the history log records a change in the issue report's attribute referring to its resolution status (this change should indicate that the issue report is either invalid or a duplicate) *Then* the user responsible for that change has performed a reviewing activity. For a *resolved* issue report, if the history log recorded a change in the issue report's attribute referring to its status, indicating that the issue report's resolution was set toverified. Then, the user responsible for that change has performed a reviewing activity. For a *closed* issue report, if the history log recorded a change in the issue report's attribute referring to its status, indicating that the issue report was reopened (i.e. status changed from closed to open). Then, the user responsible that change has performed a reviewing activity.

4.1.2.2 *User Topics Association*

Users' topics associations are represented in a $M \times T$ matrix namely $C_{U \times T}$, where U represents the total number of users, and T represents the total number of topics. $C_{U \times T}$ was formulated in a way that represents the user's level of experience within certain topics (i.e., area of expertise). The more roles (assigner, reviewer, or resolver) a user performs regarding issue reports, the more experienced he gets within topics associated with these particular issue reports. Therefore, the value of a user's u_i topics associations C_i is induced from $B_{R \times T}$ (the topic model) of the issue reports he has a role in.

$$\text{For a given } u_i \ \& \ t_j,\ C_{i,j} = \frac{\sum_1^x B_{d,j}}{x} \tag{4.1}$$

A User's u_i topic t_j association $C_{i,j}$ (shown in Equation 4.1) represents the average of topic t_j associations of all the issue reports he has a role in (where x represents the total number of issue reports the user has a role in, and $B_{d,j}$ represents the value of a given issue report r_d's topic t_i, where d represents the indices of the issue reports user u_i has a role in, the value of $B_{d,j}$ is deduced from issue reports topic association matrix $B_{R \times T}$ (the topic model)). For example, if user u_{bob} has a role in three issue reports, having associations with topic t_1 as follows: 0.03, 0.1 and 0.2; then u_{bob}'s associations of topic t_1, $C_{bob,1}$ would be the average value of topic t_1 associations belonging to the three issue reports he has a role in (i.e., 0.11 (0.03+0.1+0.2=0.33/3=0.11)).

4.1.3 *Assignee Recommendation*

After building the knowledge of all the activities performed by an issue-tracking repository's users (see subsection 4.1.2.1) and their area of expertise (topics) (see subsection 4.1.2.2), the topic model of a newly arriving issue report r_{new} is extracted. Subsequently, a list of recommended assignees corresponding to the topics of the new issue report r_{new} is created, and then ranked as per the activity profile of individual assignees. In the next subsections, we detail the process of deriving a list of recommended assignee(s) for a newly reported issue r_{new}.

4.1.3.1 *Newly Reported Issue Topics Selection*

Extracting the topic model for a newly reported issue r_{new} can be done using an online LDA method [34] in which the whole issue-tracking repository is analyzed (including the arriving stream of newly reported issue reports). However, for evaluating our approach, we did not include any in-stream data (i.e., newly arriving issue reports). Therefore, we have analyzed pre-collected issue reports (offline data) using the standard LDA method [79], where no in-stream data is considered.

After extracting a newly reported issue r_{new} topics associations, the next step is to find assignees with experience in these topics. However, we do not include all of the extracted topics associations. Rather, we select the ones that fit a certain threshold θ_j. Selecting the topics with highest significance reduces the number of candidate assignees, and only includes the most expert assignees associated with the topics. A

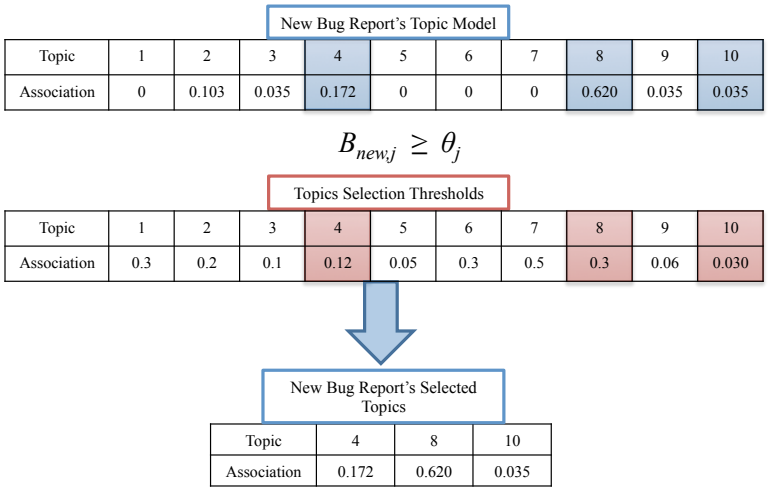

Figure 4.4: Newly Reported Issue Topics Selection

topic t_j within the n−topic model gets selected only if the newly reported issue r_{new}'s topic t_j association $B_{new,j}$ is greater than or equal to the topic t_j threshold θ_j (see Figure 4.4). Equation 4.2 defines topic t_j's threshold θ_j as the average of issue reports' topic t_j association (where R represents the total number of issue reports within the issue-tracking repository, and $B_{i,j}$ represents the value of a given issue report r_i's topic t_j association).

$$\text{For a given } t_j,\ \theta_j = \frac{\sum_{i=1}^{R} B_{i,j}}{R} \tag{4.2}$$

4.1.3.2 *Assignee Topic Matching and Elimination*

To find the candidate assignees, we consider two criteria: (1) the assignee's experience within the topics selected in the previous step (see 4.1.3.1), and (2) the number of topics the assignee should be experienced in.

The level of experience within each topic t_j is determined by the assignee u_i 's topic t_j association $C_{i,j}$ value defined in Equation 4.1. We add a constraint such that only the assignees having $C_{i,j}$ value greater than or equal to the threshold θ_j defined in Equation 4.2 are included in the list of recommended assignees.

The number of topics that the assignee should be experienced in is determined by the number topics included in the newly reported issue r_{new}. If the newly reported issue r_{new} contains more than one topic, then an assignee u_i must have enough experience in at least half of the topics to be included in the candidate list. If the new issue report r_{new} has only one topic, then the assignee u_i with the required level of experience in this topic is added to the candidate list. We use the criteria of satisfying at least half of the topics in order to provide sufficient scope (so that all relevant developers are on the candidate list, and still, not everyone). This criteria can be changed to consider any assignee satisfying all the required number of topics within the new issue report r_{new}, but this will restrict many relevant (though not so experienced (in all topics) assignees) from getting onto the candidate list.

4.1.4 *Assignee Ranking*

To be able to rank the list of the new issue report r_{new}'s matched assignees, we needed to compute a ranking score RS for each of them. The assignee u_i's RS_i is deduced from both his activity profile and the new issue report r_{new}'s topic model. RS_i represents the assignee u_i's capability of resolving the issue report (which is determined by his topic expertise and issue-tracking activities).

$$\text{For a given } u_i,\ RS_i = \sum_{1}^{y} (C_{i,g} \times B_{new,g} \times Score_g) \tag{4.3}$$

RS is defined in Equation 4.3, where y represents the total number of topics within the new issue report r_{new} that is needed to be matched with assignees' expertise, $C_{i,g}$ represents the assignee u_i's topic t_g association (i.e., level of topic expertise (from Equation 4.1)), $B_{new,g}$ represents the new issue report's topic t_g association (g represents the value of indices referring to the selected topics within r_{new}), and $Score_g$ represents the assignee u_i's issue-tracking activities score within a topic t_g (from Equation 4.4).

The activity score is considered a key element in ranking the assignees, since it induces the frequency and type of activities performed by the assignees within a certain topic, thus indicating the assignee's role. The issue-tracking activities' frequencies differ within the same project, and that is why the activity score was formulated in a way to favor certain activities over others (for the purpose of favoring the assignees engaged in performing the most frequent activities, since they have the most influential roles and expertise within the project). In Equation 4.4, k represents the total number of issue-tracking activities (in this case, the three aforementioned activities we mine from the user's role described in Section 4.1.2.1).

$$\text{For a given } t_i \ \&\ u_L,\ Score_i = \sum_{k=1}^{3} w_k \times \frac{A_{k,i}}{A_i} \tag{4.4}$$

For each activities score $Score_i$ of a given topic t_i and assignee u_L, $A_{k,i}$ represents the number of times an assignee has performed an activity a_k within a specific topic t_i. A_i represents the total number of issue-tracking activities related to a specific topic t_i and performed by a specific assignee u_L. We added a constraint that limits the number of issue reports counted within $A_{k,i}$ and A_i, to the ones having topic t_i association value that is either greater than or equal to topic t_i's threshold θ_i defined in Equation 4.2.

$$\textit{For a given } a_k,\ w_k = 10 \times \frac{Z_k}{A} \tag{4.5}$$

The variable w_k represents the weight of a certain activity a_k. w_k acts as a tuning parameter defined in a way that gives a high activity score $Score_i$ for assignees engaged in the most common activities (i.e., most influential roles). w_k is a real number that varies from 0.0 to 10.0 depending on the activity a_k's frequency (compared to all other issue-tracking activities) as shown in Equation 4.5. The value of w_k is deduced from

the occurrences of activity a_k, where Z_k represents the total number of occurrences of a_k, and A represents the total number of all issue-tracking activities performed within the issue-tracking repository. Consequently, issue-tracking activities with high rate of occurrences will have higher activity weight w_k.

4.2 CASE STUDY

In this section we describe the research strategy, which investigates the use of *INExPERT* within a scientific software project. Our main hypothesis in this study is that "integrating *INExPERT* within the issue-tracking practices of scientific software projects, will result in an increase of issue assignment quality in terms of: (1) reducing the number of unassigned issue reports and (2) increasing the probability of selecting experienced assignees". This was evaluated using qualitative and quantitative data collected by performing a *pre-post quasi experiment* and a *structured interview*.

The first of our study goals was to validate the correctness of *INExPERT's* recommendation model. The second goal was to prove *INExPERT*'s usefulness for scientific software projects. The third was to identify the problems and shortcomings of *INExPERT's* preliminary design pointing out any missing requirements and/or domain specific needs.

Section 4.2.1 describes that the study has followed a four-stage formative evaluation approach involving the development of a rapid prototype that is further detailed in Section 4.2.2.1. Throughout the formative evaluation approach, the prototype was tested by a group of scientific software domain experts involved in a mid-sized scientific software project serving an experiment within the particle physics domain. The experts' responses towards the use of the prototype were captured through a pre-post quasi experiment and a structured interview that are further discussed in Section 4.2.2 and Section 4.2.2.3. Collecting the experts' responses from the targeted environment helped us in gaining a rapid feedback and collecting their ideas about how could the implementation of *INExPERT* be improved.

4.2.1 *Formative Evaluation*

Formative Evaluation [71] takes a social science perspective to evaluating how to enhance a specific process prior to its completion. However, in software development, the formative evaluation approach is used as a software examination method to obtain feedback for several purposes, such as: (1) refining software development's scope, and (2) improving aspects of software design like functionality or usability. Consequently, formative evaluation consists of several activities involving the collection and analysis of project data and users' or development team's feedback. It incorporates some techniques, such as in-depth interviews, surveys, observations, and dialogues with participants. Accordingly, findings from these evaluation activities can help identify potential areas of improvement within the software development process through investigating the changes from targeted baselines.

Figure 4.5 shows that the case study has followed a formative evaluation approach consisting of four stages: briefing, pre-*INExPERT*, post-*INExPERT* and rationale & feedback.

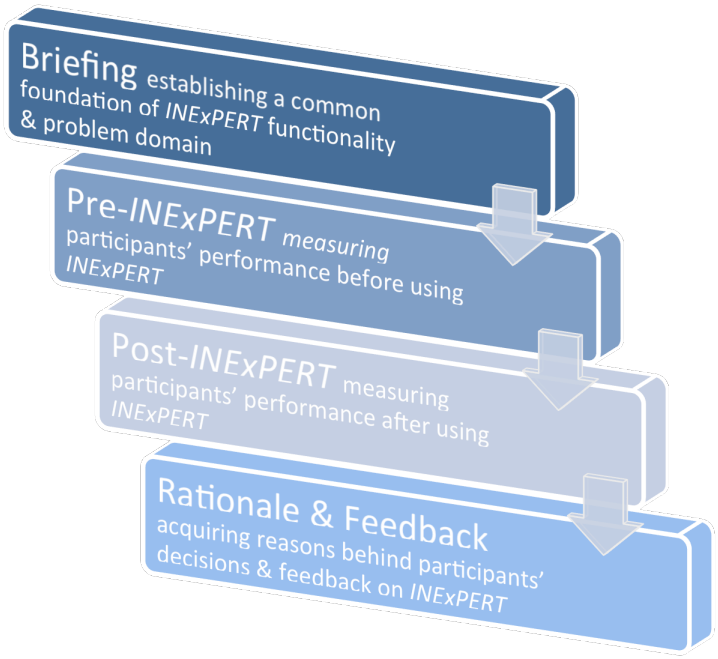

Figure 4.5: Formative evaluation stages

These stages are described in detail within Sections 4.2.1.1 till 4.2.1.4. The case study was conducted by applying a pre-post experiment design described in Section 4.2.2. The experiment involved a group of seven experts within scientific software projects. Data was collected during the time when participants have already manually assigned a group of issue reports. The same data was once again collected after the participants have assigned the same group of issue reports using the *INExPERT* prototype. To that end, we have used a web interface described in Section 4.2.2.2, that applies a mock object of the *INExPERT* prototype. The mock object mimics *INExPERT*'s behavior in predefined scenarios. By using the web interface with its different functionalities, we were able to observe, interview, and record activities of the participants during controlled experiment scenarios.

After recording the participants' behavior and responses to the use of the *INExPERT* prototype, we analyzed and compared the data recorded during both phases of the experiment (i.e., pre- and post-*INExPERT*). This aimed at validating *INExPERT*'s precision in terms of (1) recommendations and ranking of assignees and (2) heuristic rules used in describing issue-tracking activities. The evaluation also aimed at investigating whether *INExPERT* is suitable to be used within scientific software projects in general. To this end, we collected and analyzed the experts' responses on how to improve the current design to make it more suitable and effective in meeting the scientific software domain's requirements.

All four evaluation stages were recorded via a screen- and audio-recording software to obtain a more accurate documentation of the evaluation activities, inquiries, and feedback of each participant. This reduced the vulnerabilities of lost/misunderstood information and helped achieve accurate analysis and review of the experiment's outcomes and interpretations. This also gave more freedom for the researcher to think,

interact and engage with the participants without worrying about tracking what was happening. Additionally, going through each recorded evaluation session provided us with more insight into whether they were performed in the intended conduct.

In conclusion, the outcome from this formative evaluation activities was used to identify new requirements and improvements that were later included in the formal design and implementation of *INExPERT*. We actually discarded the rapid prototype afterwards.

4.2.1.1 *Briefing Stage*

This stage is designed to last for 10 to 15 minutes. The briefing stage is used to ensure that all participants have the common foundation regarding *INExPERT*'s functionality and the problem's domain, aiming at reducing misconceptions and misinterpretations of the goals and procedures of the experiment. Towards that end, the researcher uses different presentation aids to provide the participants with an overview about (1) the motivation behind the conducted research work, (2) the experiment's goals, (3) the scope of this work within the issue report life cycle, (4) the different types of issue assignment techniques, (5) the tasks he has to perform, (6) issue-tracking activities, and (7) *INExPERT*'s functionality.

4.2.1.2 *Pre-INExPERT Stage*

This stage serves as a **baseline** to identify the participants' decisions regarding every issue report they receive.

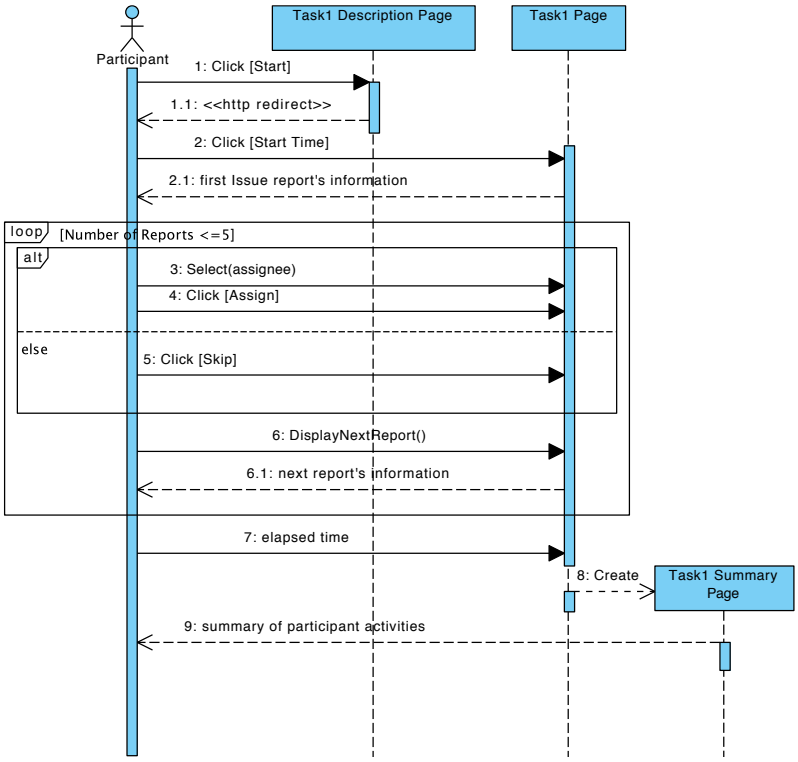

Figure 4.6: Pre-INExPERT Stage's Activities

We have recorded the participants' decisions **before** and **after** using the aid of *INExPERT*, which allowed us to draw inferences on the effect of using *INExPERT* on assigning issue reports (by observing what changed from the baseline). In more details, defining the differences between "Pre" and "Post" stages in terms of (1) the rate of assigned reports, (2) selected assignees and (3) their relevance to the list of recommended assignees.

First the researcher initiates the Pre-*INExPERT* stage, as illustrated in Figure 4.6. Each participant is presented with a brief description of an assignment task that he needs to perform (i.e., a new issue report waiting for being assigned). Every participant manually assigns five issue reports to project members they think are capable of handling. The issue reports that were given to the participants were previously assigned, resolved and closed. This was done to ensure that the involved issue reports had already completed their life cycle, thus insuring a complete and consistent content. Additionally, in order to standardize our findings, all participants got to assign the same set of issue reports. When a participant was ready to perform the assignment task, she could click *Start*. That directed her to the webpage presenting the issue reports. She then was asked to initiate a timer by clicking *Start Time*, in order to record the time she took to successfully complete this stage. For every issue report, she was asked to **either** (1) select a specific assignee from a given list of project members then click *Assign* to submit her decision **or** (2) click *Skip* in case she was unable to determine a suitable assignee. Afterwards, she was redirected to a page containing a summary of outcomes (**selected assignees** and the **time** it took her to do the whole round).

4.2.1.3 *Post-INExPERT Stage*

This stage helped us identify changes to the baseline (i.e., participants' decisions obtained in the Pre-*INExPERT* stage). It clarified how our participants behaved when they received a list of assignee recommendations from *INExPERT* (i.e., changed their decision or not). Additionally, it helped us gain a deeper insight into the "satisfaction index" of the involved participants (how satisfied and motivated they were to use *INExPERT* when assigning new issue reports). To that end, we asked the participants to rate the quality of the generated assignee recommendations in terms of (1) how helpful the list was in reaching a decision (helpful), (2) whether the list contained suitable assignees (complete), and (3) how accurately the list was ranked (accurate).

The stage included many activities (shown in Figure 4.7). First, a participant was presented with a brief description of an assignment task, in which he was required to assign five issue reports guided by a list of recommended assignees generated by *INExPERT*. After clicking *Start*, the webpage presenting the task popped up. He then clicked *Start Time* to record the time he took to successfully complete this stage (note the similarities to the Pre-*INExPERT* stage). For each issue report, he was asked to **either** (1) select a specific assignee from the list of recommended assignees, rate the quality of the generated list, and click *Assign* to submit his decision and evaluation **or** (2) click *Skip* in case he was unable to determine a suitable assignee. Upon the successful completion of this task, he was redirected to a page containing the summary of changes compared to the baseline.

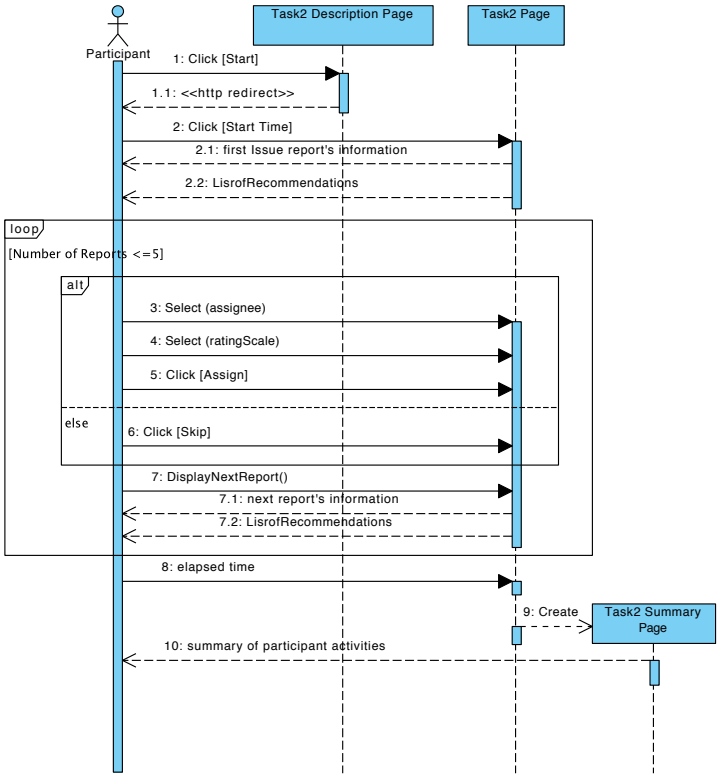

Figure 4.7: Post-INExPERT Stage's Activities

4.2.1.4 *Rationale & Feedback Stage*

This stage was designed to last for 15 to 20 minutes, where we interviewed each participant to (1) understand the rationale behind their activities during the Pre- and Post-*INExPERT* stages and (2) acquire their impressions about possible shortcomings of *INExPERT*'s design, drawbacks, and potential impacts of integrating it with the project's current issue-tracking practices. To that end, during the interview we focused on identifying the following:

- The similarities/differences between the rationale of
  - manually choosing an assignee
  - choosing an assignee guided by *INExPERT*
- The shortcomings of the heuristic rules implemented to define issue-tracking activities.
- The expected drawbacks of integrating *INExPERT* within their project's issue-tracking practices.
- The expected impact of using *INExPERT* on the average resolution time of issue reports.
- The degree of issue assignment automation (semi vs. full) preferred by most participants.

The interview followed a structured layout (mentioned in Section 4.2.2.3) in which all participants were asked the same questions within the same order and answering options, enabling us to reliably aggregate and compare all the responses with a high confidence.

4.2.2 *Pre-Post testing Quasi-Experiment*

By using a Pre-Post testing experiment design, we gathered data on the participants' decisions when they were asked to assign a group of issue reports without the use of *INExPERT* (pre-), and did the same after they used *INExPERT* (post-) as shown in Figure 4.8.

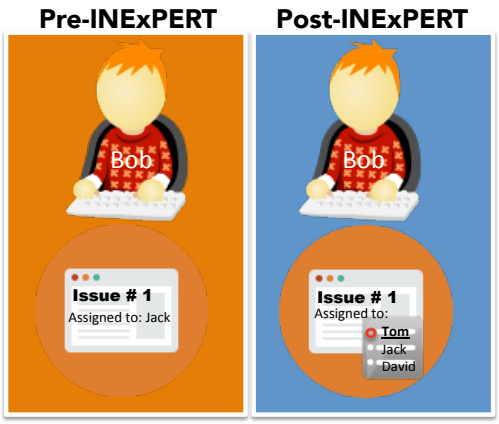

Figure 4.8: Comparison of a participant's decision before and after the use of INExPERT

This type of experiment design focuses on one group of individuals who have received the intervention (i.e., *INExPERT*), in which the collected data is compared for providing an explanation of the effects of the used intervention on each participant. By collecting the decision of each participant twice (before and after the intervention), we have paired observations of the same subject. The whole point of using these paired observations is to control the experimental variability as each subject serves as his own control group, since we only measure the differences among the same subject and thus reduce the variability that can occur between different subjects. However, the sample has to be doubled to achieve the same number of degrees of freedom as unpaired samples.

To give the participants a realistic view of how *INExPERT* would look like if it were integrated into their current issue-tracking practices, we constructed a web interface that simulates the integration of *INExPERT*'s prototype with their issue-tracking system (it was not possible to apply it to the actual system, due to time and managerial constraints). Section 4.2.2.1 and Section 4.2.2.2 describe in detail how *INExPERT*'s prototype and web interface were implemented. Finally, we interviewed the participants to clarify the reasoning behind their performed activities during the experiment stages. The interview followed a structured layout described in Section 4.2.2.3 to achieve more accurate interpretations of the outcomes.

Since the main focus of our research is to identify and analyze the impacts of using *INExPERT* within software development teams of scientific software projects, we conducted the experiment within an ongoing project of the Munich Centre of Advanced Computing: MAC-B2[4], which attempts to support and enhance the current issue-tracking practices of a scientific software project called ATLAS-Reconstruction[5] in a way that reduces the time and effort exerted. ATLAS-Reconstruction is a part of the software infrastructure of a particle accelerator experiment conducted at the Large Hadron Collider (LHC) at CERN in Geneva, Switzerland. Its software infrastructure team wrote about 7 million lines of code for event reconstruction in their experiments.

We conducted the experiment within the first two weeks of December 2012, during which we consulted the software project leader of ATLAS-Reconstruction to find participants with diverse roles and years of experience. The project leader selected 15 project members out of 52, based on their background and availability by that time. We contacted all 15 project members via email to explain the purpose of the experiment, its procedures, and its duration. Only seven project members agreed to participate, which naturally reduces the observations' level of significance. Table 4.2 gives a summary on the background and years of experience of the participants.

Table 4.2: Experiment Participants

Project Role	**# of participants**	**Years of Experience**	
Senior Convener	3	Average	6
Software Librarian	1	Minimum	2.5
Senior Developer	2	Maximum	10
Project Leader	1	Standard Deviation	2.66

4.2.2.1 *Rapid Prototyping of INExPERT*

In this section we describe how we have implemented a rapid vertical prototype of *INExPERT* to help experts understand how the recommendation technique is intended to work. The sole purpose of this prototype was to gather their feedback regarding the shortcomings of the initial design of *INExPERT* in terms of (1) how it identifies different issue activities, (2) how it is intended to match and eliminate the assignees, and (3) how it ranks the list of recommended assignees. Based on their feedback, we were able to identify points of potential improvement that might help achieve more accurate and correct outcomes. After that, the prototype was completely discarded and *INExPERT* was formally developed based on the identified points of potential improvement. Addressing these at an early stage of development was very effective, as it helped reduce the amount of rework needed for improving the technique's performance later.

The prototype was implemented in Java. Figure 4.9 describes an abstraction of its design. The prototype is represented as a composition of both (1) a **FeatureExtractor** responsible for extracting the relevant information regarding topics associated with the involved issue report and the relevant project members, and (2) an **RAListGener-**

4 http://www.mac.tum.de/wiki/index.php/Project_B2-b2
5 http://atlas.ch/

ator responsible for generating a list of ranked recommended assignees for a specific issue report.

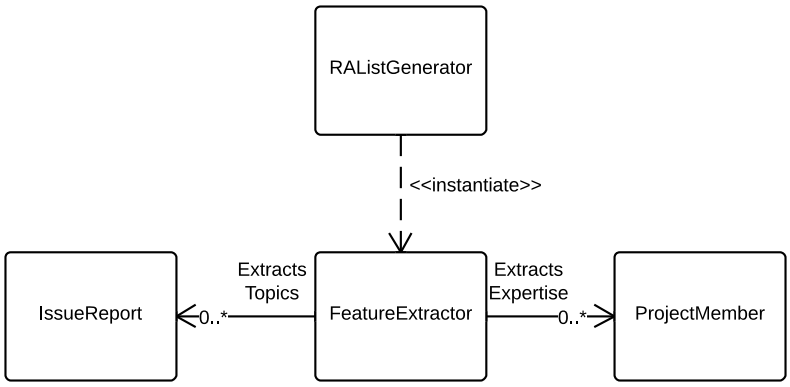

Figure 4.9: INExPERT's Early Design Abstraction

FeatureExtractor is basically responsible for retrieving the data related to the topics of issue reports and the expertise of project members, through a set of implemented queries. In order to be usable by machine learning techniques, this data is converted into numerical values and stored in a unified database. Details on how these features are converted and stored were previously mentioned in Section 4.1.1 and Section 4.1.2.

RAListGenerator is responsible for matching, eliminating and ranking assignees. Figure 4.10 illustrates an overview of the activities involved in generating a ranked list of assignees that may be able to resolve a specific issue report.

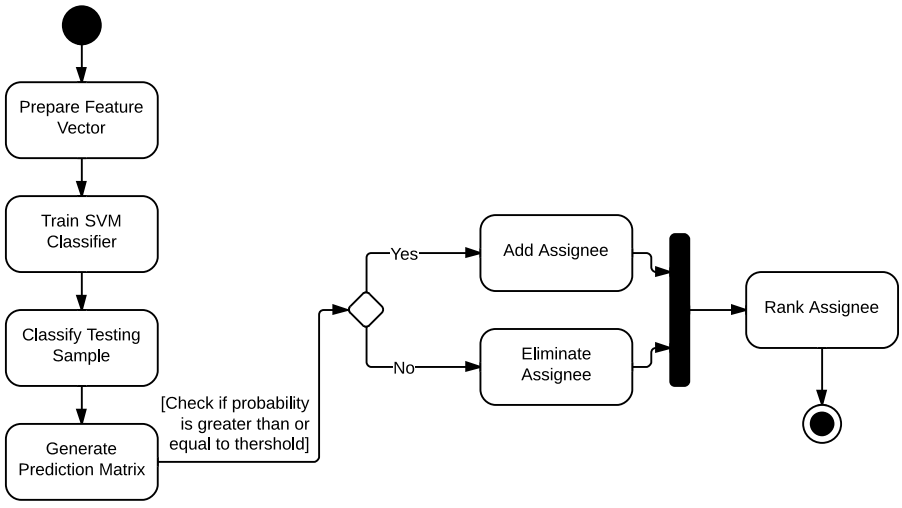

Figure 4.10: Overview of the activities done to generate and rank a list of recommendations

To match assignees for a specific issue report we use SVM. The implementation of SVM was provided by Waikato Environment for Knowledge Analysis (Weka) [28], a machine learning API. SVM analyzes the features provided by **FeatureExtractor** and recognizes certain patterns related to the topics associated within each issue report. It then matches these patterns to a specific assignee (class or label). Features extracted from each issue report form a model that represents topics associations as a $R \times T$

matrix, where R represents the total number of issue reports and T represents the total number of topics. Each column in a given row $M_{i,j}$ contains a fraction of the value of the topic association of $report_i$ with $topic_j$ against the rest of values in M_i (more details were mentioned in Section 4.1.1). Given the $M_{R\times T}$ matrix (with each row belonging to one assignee), the SVM training algorithm builds a model that classifies every set of features to a class (in our case, assignee), rendering it a non-probabilistic binary linear classifier. The features of the training set are represented as points in an N-dimensional space whose mapping tries to assure that issue reports belonging to separate classes/assignees are divided by a clear hyperplane that is as wide as possible. New issue reports (i.e., the testing sample) are then mapped into that same partitioned space, and are predicted to belong to a certain assignee based on which portion of the space they fall in.

This prototype uses the Sequential Minimal Optimization (SMO) [62] multi-class classifier provided by Weka, since it is one of the most commonly used classifiers to solve multi-class problems. It creates multiple hyperplanes to split the points among the classes/assignees. The classifier tries to identify the assignee of an issue report among all possible pairs of assignees (one-versus-one). The classification is done by a max-wins voting strategy [20], in which the classifier matches the issue report to one of the two assignees in the pair. The appointed assignee's score is increased by one vote, and the assignee with the highest score determines the classification. After building the SVM training model that indicates how issue reports are mapped to project members' area of expertise (topics), a list of recommended assignees corresponding to the topics of the new issue report (i.e. testing sample) should be created. However, as previously mentioned, SVM matches only a single assignee, the one having the highest voting score. Consequently, the generated list will contain only one recommendation. Assignees having high probability yet scored a lower voting score than the classified assignee are still likely to posses many of the desired levels of expertise in one or most of the required topics. Nevertheless, SVM discards them from the classification output. This will defiantly exclude assignees that are capable either of reassigning or resolving the issue. In order to include these assignees within the generated list of recommendations, we have used the prediction metrics reported by Weka. The prediction matrix contains the probability that each issue report actually belongs to an assignee. From the matrix, we were able to identify the predicted assignee and the probability distribution among all other assignees for each issue report. Subsequently, a list of recommendations that includes assignees having a probability value less than or equal to the chosen assignee (i.e., maximum level of expertise) and greater than or equal to a specific threshold (i.e., minimum level of expertise) can be generated. The threshold is defined at runtime (i.e., considered to be a tuning parameter).

Finally, the ranking phase. To be able to rank the list of recommended assignees, we need to compute a ranking score RS for each of them. Assignee i's RS_i is deduced from both (1) the probability value assigned to them (by Weka's prediction matrix, details in the previous paragraph) and (2) the type and frequency of all activities they have performed. RS is defined in Equation 4.6, where $Assignee_iProbability$ represents the probability of classification assigned to $assignee_i$ which indicates their

capability of resolving the issue, and $ActivityScore_i$ represents $assignee_i$'s issue-tracking activities score.

$$\text{For a given } assignee_i,\ RS_i = Assignee_i Probability \times 100 + ActivityScore_i \quad (4.6)$$

The activity score $ActivityScore$ is considered a key element in the process of ranking assignees, since it induces the type and frequency of activities performed by each assignee (thus indicating their role). $ActivityScore$ was formulated in such a way to favor certain activities over others for the purpose of selecting assignees engaged in performing the most influential roles, such as resolver and manager roles. An assignee belonging to either of these two roles is experienced enough to handle the given issue as he has formerly done that (i.e. resolved or managed issue reports). $ActivityScore$ is defined in Equation 4.7, where w_k represents the different weights of the four activities related to the most knowledgeable assignees.

$$\text{For a given } assignee_i,\ ActivityScore_i = \sum_{k=1}^{4} w_k \times \%activty_k \quad (4.7)$$

These four activities are (1) prioritizing, (2) assigning, (3) reviewing and (4) resolving. w_k is an integer number that varies from 1 to 4, and was set to favor the resolving activity (four, the highest number) over the other three. The prioritizing activity gets the lowest value (one) as it is often hard to indicate if the person is directly involved in any managing tasks. $\%activty_k$ represents the percentage of performing an $activity_k$ over the total number of all issue-tracking activities performed by $assignee_i$. Consequently, assignees having a high rate of performing "high weight" activities will have higher $ActivityScore$ over others assignees, thus giving a higher rank to more knowledgeable assignees.

In general, an experiment's participants tend to be motivated in putting more effort and thought into the study if the experiment were set in a realistic context that is connected to their organizational settings or professorial experience [7]. Accordingly, we created a web interface that demonstrates a broad view of how it would look like if *INExPERT* were integrated into their current issue-tracking practices, as shown in Figure 4.11.

4.2.2.2 *INExPERT's Web Interface*

As we can see in Figure 4.11, the details of the issue report were laid out along with the list of recommended assignees. This setting can help the participants have a better understanding of (1) how *INExPERT* is intended to function and (2) its role within the issue-tracking practices.

The web interface implemented applies a mock object pattern [43] to imitate the behavior of generating *INExPERT*'s list of recommended assignees. Instead of providing real *INExPERT* functionality, the web interface was used to capture the participants' interactions and feedback during the experiment. Each participant was asked for their feedback on how useful and accurate the given list of assignee recommendations was. The feedback was captured through an evaluation form shown on the bottom of page

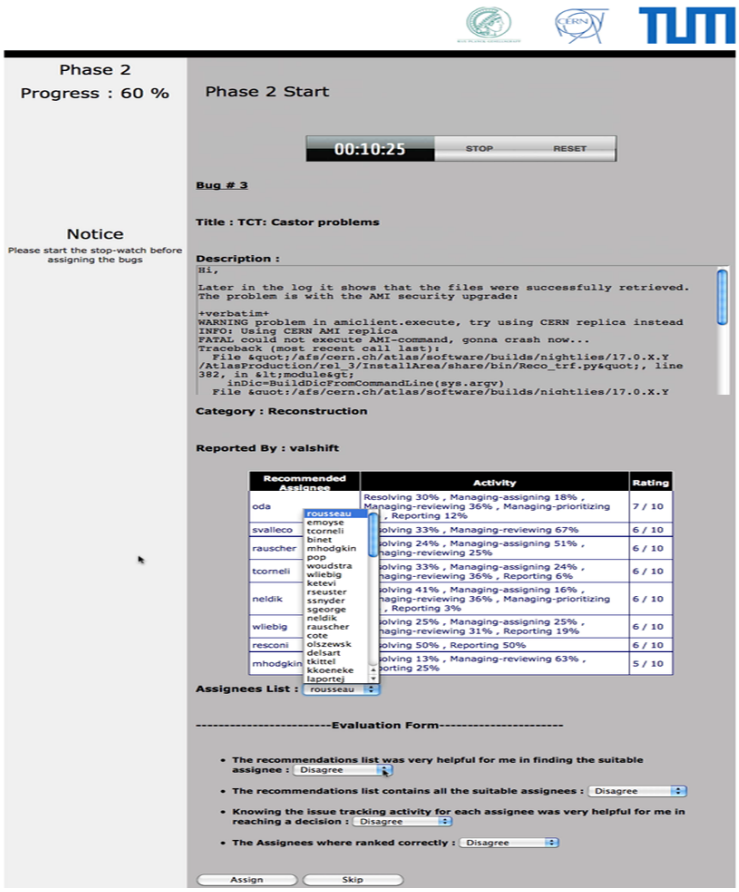

Figure 4.11: Overview of INExPERT integrated into issue-tracking practices

(see Figure 4.11). Finally, each stage within the experiment was timed using a timer (located at the upper side of page, see Figure 4.11).

The web interface was implemented using a Three-Tier architecture as shown in Figure 4.12. It was developed using ASP.NET and a MySQL database. The first package represents the **Presentation Tier**, which is responsible for displaying information related to the experiment stages. It involves (1) the *BriefingStagePage*; representing the output that introduces the main goals of the experiment to the participant, and collects their personal information (project role and years of experience), (2) the *Pre-INExPERTStage-Page*; representing the user interface displaying manual assignment tasks (in which data is collected on participants' interactions), (3) the *Post-INExPERTStage-Page*; representing the user interface used to collect data on participants' interactions and feedback after introducing *INExPERT*, and (4) the *FeedbackStage-Page*; representing a summary on participants' interactions during the Pre-Post *INExPERT* Stages. Additionally, it displays the interview questions that were used to collect more information about the participants' interactions and feedback during the experiment. The second package is the **Application Tier** that is represented as a black box for simplification. It consists of a lot of detailed classes that are responsible for controlling and updating the changes within the Presentation Tier and the Data Tier. The final package represents the **Data Tier**.

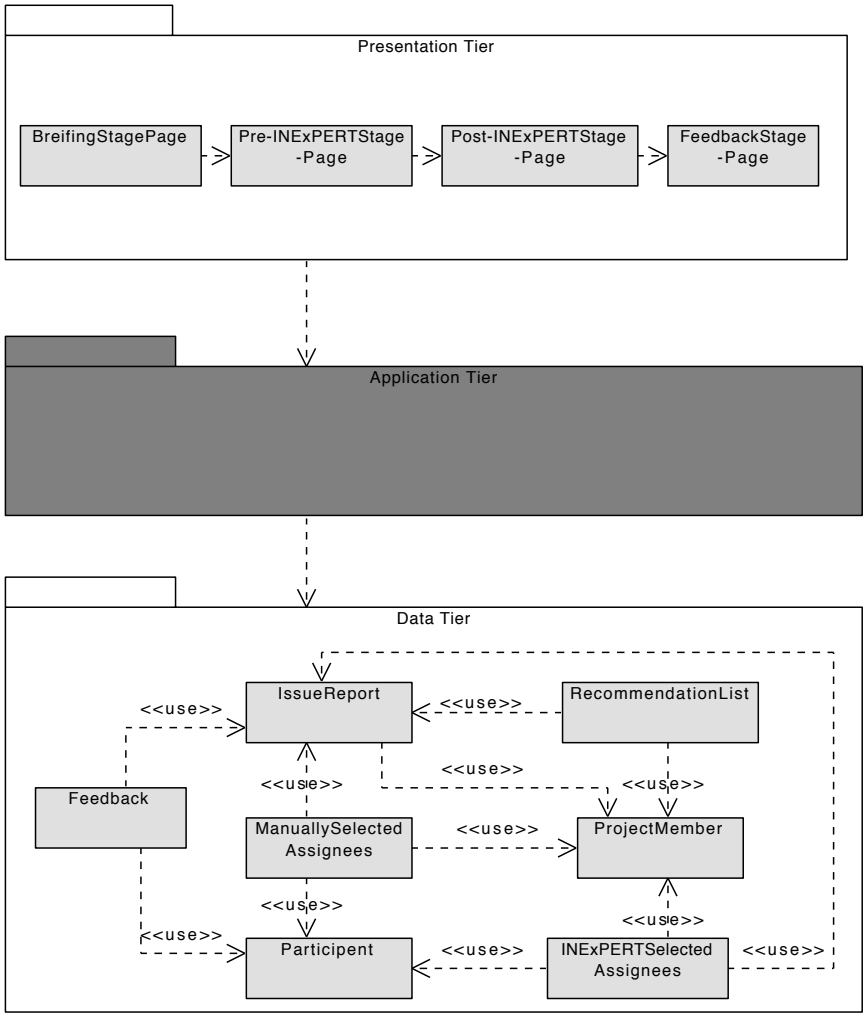

Figure 4.12: Overview of the Website Architecture

The **Data Tier** package notifies the **Application Tier** classes after changes (resulting from changes in the **Presentation Tier**) have occurred in the database. The package consists of seven classes interacting with the database. The database contains information related to the experiment stages and participants' interactions and feedback. The **IssueReport** class is responsible for presenting details about issue reports (such as a report's title, description and reporter). The **RecommendationList** class is responsible for conveying the list of recommended assignees that was generated by *INExPERT*. Since we implemented a mock object pattern, we don't intend to provide any real system functionality for *INExPERT*; that's why both classes were set to interact with static data generated from *INExPERT*'s prototype (details in Section 4.2.2.1). First, we randomly selected 20 issue reports from the issue-tracking repository of ATLAS-Reconstruction (only from the project's last 12 months). We then divided these issue reports equally into (1) a training set and (2) a testing set. We used both data sets against *INExPERT*'s prototype and generated lists of recommended assignees for the 10 issue reports in the testing set; from which we randomly selected 5 issue reports

along with their list of recommended assignees to be added to the database used within the experiment. Furthermore, we added the **ProjectMember** class and the **Participant** class responsible for retrieving and updating the details of project members and participants (name, role, etc...). **ManuallySelectedAssignees** and **INExPERTSelectedAssignees** classes were responsible for retrieving and updating the database with participants' decisions regarding choosing an assignee for each issue report before and after using *INExPERT*. The **Feedback** class updates the database with the participants' feedback on how helpful and accurate the list of recommendations generated by *INExPERT* was.

4.2.2.3 *Interview Outline*

In order to get a more clear insight into the rationale behind the decisions made by the participants during the different stages of the experiment, we developed an outline of a structured interview that aimed at gathering information on the impact imposed by using *INExPERT* on the participants' choices. Additionally, the interview was designed to gather the participants' feedback on the formulation of the heuristics that define the issue-tracking activities; as these may significantly influence the accuracy of *INExPERT*'s outcomes.

The interview layout consisted of five open-ended and three close-ended questions. Two open-ended questions asked the participants to state the main aspects they considered while choosing an assignee, once without using *INExPERT*, and once again while using it. Additionally, two close-ended questions asked the participants to answer with a yes/no, stating the reasons of (1) whether using *INExPERT* would reduce issue reports' resolution time and (2) whether they preferred an automated process of issue reports' assignment. Furthermore, two open-ended questions asked participants to state (1) the drawbacks of integrating *INExPERT* into their current issue-tracking practices and (2) the reasons behind having unassigned issue reports that were closed. Another close-ended question evaluated the level of (dis)agreement the participants had on whether issue-tracking activities do affect the ranking of assignees within *INExPERT*'s list of recommendations, in which we used a five-level symmetric (dis)agree scale. Finally, we created a graphical and textual description of the heuristic that we formulated to describe the issue-tracking activities, which was then added to an open-ended question that asked the participants to state whether these heuristics where formulated correctly, in addition to justifying their answers. A copy of the interview's layout is described in the Appendix at Section A.1.

4.2.3 *Experiment Metrics and Findings*

Section 4.2.3.1 gives a brief description of the experiment's metrics and their calculation methods. These metrics were used to compare participants' decisions before and after using *INExPERT*. Section 4.2.3.2 analyzes and highlights the testing results of the experiment's main hypothesis: Integrating *INExPERT* within the issue-tracking practices of scientific software projects will improve issue assignment quality in terms of (1) reducing the number of unassigned issue reports and (2) increasing the probability of selecting experienced assignees.

4.2.3.1 *Experiment Metrics*

Table 4.3: Experiment Metrics

Metric	Description	Purpose	Metric Type
Number of Assigned Issues	The number of issue reports assigned by each participant during a specific stage	Evaluates whether using *INExPERT* reduces the number of unassigned issue reports	Quantitative
Recommended Assignee Hit Ratio	A "hit" occurs when the chosen assignee during a specific stage belongs to *INExPERT*'s list of recommended assignees	Investigates whether using *INExPERT* increases the probability of selecting experienced assignees	
Number of Matched Choices	Summarizes the outcomes of the Pre- and Post-*INExPERT* stages by indicating the number of times the chosen assignee was the same within both stages	Assesses "learning effects" among participants due to the order of presentation of issue reports	
Helpful	An ordinal scale that indicates how helpful the list of recommended assignees is for finding a suitable assignee	Evaluates how beneficial *INExPERT* is to the experts	Qualitative
Complete	An ordinal scale that indicates wether the list of recommended assignees contains suitable assignees	Evaluates the experts' satisfaction with finding suitable assignees within the list of recommendations	
Accurate	An ordinal scale that indicates how accurately the list of recommended assignees was ranked	Evaluates the experts' satisfaction with the accuracy of the list of recommendations ranking	

Within this experiment, we used six metrics, described in details in Table 4.3. Since the experiment focused on paired observations per subject (to identify the impact of using *INExPERT* on the number of unassigned issue reports and the probability of selecting experienced assignees), the first two quantitative metrics were adopted; namely (1) **Number of Assigned Issues** and (2) **Recommended Assignee Hit Ratio**. These were measured twice; before and after the subject used *INExPERT*.

Since participants got to assign issue reports manually before assigning the same ones using *INExPERT*, they might have developed enough familiarity with the presented issue reports in a way that could influence their choices in the second time. We added additional steps to account for those learning effects, such as changing the order of the presented issue reports in the second time, and placing the list of recommendations before the UI components that captured the participants' choices (in order to "force" the participant into reading the presented recommendations before rushing into doing the same action as the first time). It was vital to identify whether the approach used to compensate for learning effects was effective enough, so we adopted the **Number of Matched Choices** metric that simply counted the number of times the chosen assignee was the same within the pre- and post- *INExPERT* stages. To evaluate and understand the experts' impressions on how helpful and accurate *INExPERT*'s list of recommendations was, we adopted three qualitative metrics. Refer to Table 4.3 for details.

4.2.3.2 *Experiment Findings and Interpretations*

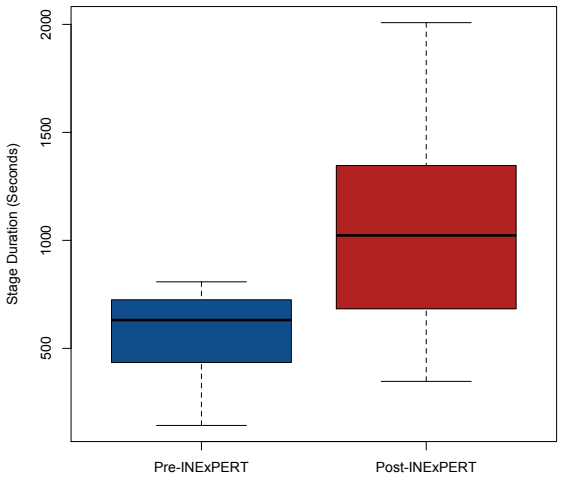

Figure 4.13: Stage's Duration

Figure 4.13 indicates that the participants spent more time engaging within the Post-*INExPERT* Stage over the Pre-*INExPERT* Stage; since they had several inquires regarding the layout, role and ranking of the recommendations generated from *INExPERT*. Since our sample size was very small (less than 10 participants contributed to the experiment), it is unlikely that normality tests will detect any non-normality. Nevertheless, we used one parametric test —the **Paired T test** [88] —to test our interval and ratio metrics (based on the assumption that our population distribution is normal). Additionally, we used a non parametric test —**Fleiss' Kappa** [23] —that is more suitable for testing our ordinal scale metrics (in which the level of agreement between participants was induced).

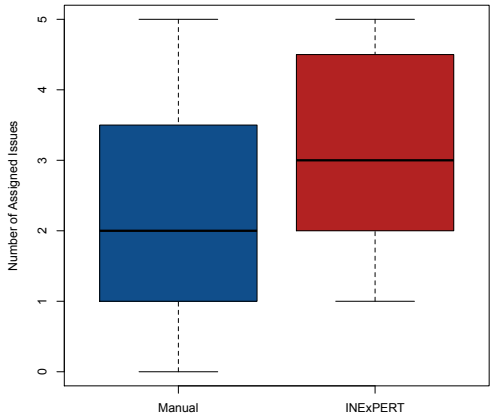

Figure 4.14: Number of Assigned Issues (Manual Vs. INExPERT)

Figure 4.14 indicates that the number of assigned issue reports tends to increase (or remain the same) for most of the participants after using *INExPERT*. We used a paired t-test, having t = -1.6859, 6 d.f., and p-value = 0.071. Unfortunately, due to the small sample size, the increase in the number of assigned reports couldn't be significantly indicated through the t-test.

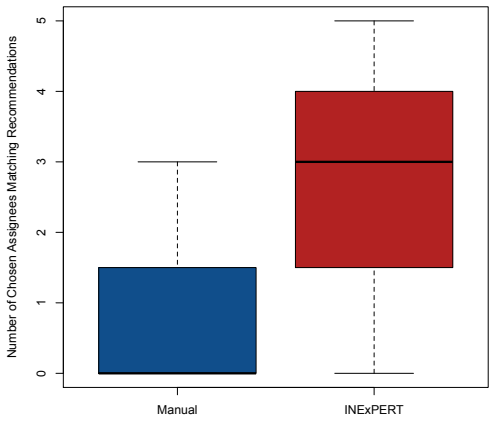

Figure 4.15: Number of Chosen Assignees Matching Recommendations (Manual Vs. INExPERT)

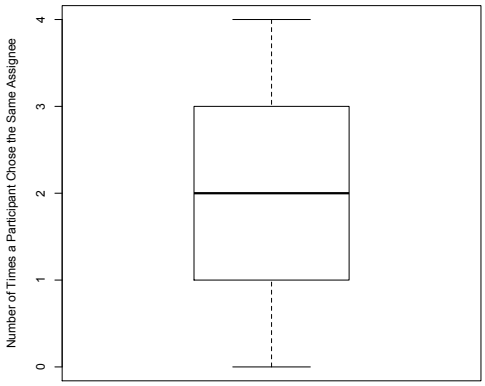

Figure 4.16: Number of Times a Participant Chose Same Assignee Within Pre- & Post-*INExPERT* Stages

Figure 4.15 shows that during the manual assignment task, some participants did choose assignees that were later included in the list of recommendations. Subsequently, using *INExPERT* did increase the number of chosen assignees that belonged to the list of recommendations. We used a paired t-test to validate this increase, having t = -2.9314, 6 d.f., and p-value = 0.013, which resulted in supporting our hypothesis that using *INExPERT* increases the probability of selecting experienced assignees.

The main point of Figure 4.16 is to illustrate the existence of learning effects, through the number of times a participant chose the same assignee within both Pre- & Post-*INExPERT* stages. We conducted a one sample t-test to investigate whether participants went for same choices within both stages. Using t = -5.1962, 6 d.f., and p-value = 0.002, the test validated the effectiveness of the approach we used to compensate for learning effects.

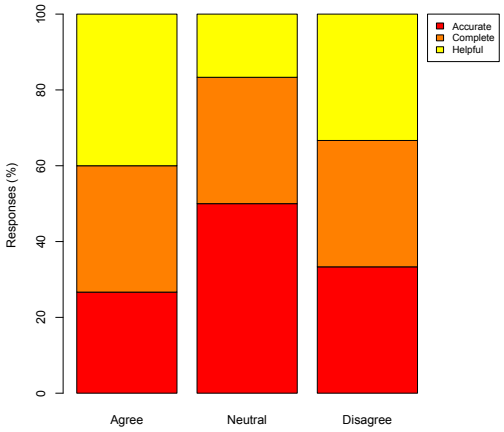

Figure 4.17: Responses Vs. Qualitative Metrics

We used three qualitative metrics to determine how helpful, complete and accurate *INExPERT*'s recommendations were to the participants during the assignment of all given issue reports. Figure 4.17 indicates that the participants' responses stated that the list of recommendations were within 26.67% of the time accurate, 33.33% of the time complete and 40% of the time helpful. To measure inter-rater reliability (i.e., the degree of agreement between participants' ratings concerning each issue report), we used a Fleiss' Kappa test; which indicated that the participants' ratings were mostly heterogeneous. Having only a slight-to-fair agreement among all raters on the list of recommendations being helpful with a Kappa = 0.2 and p-value = 0.045.

4.2.3.3 *Interview Findings and Interpretations*

All seven experts mainly considered the relevance of the assignee's expertise and responsibility to the involved package or sub-group when they chose an assignee (both manually and using *INExPERT*). One difference while using *INExPERT* was taking into account how the ranking was associated with the assignee's responsibility within the package. To that end, we had to modify *INExPERT*'s design to consider the association between the assignee's level of expertise within specific packages or topics and his responsibility within these topics (i.e., role (main developer, package owner, etc...)).

We asked the experts to identify the shortcomings in the heuristic rules we implemented for defining the issue-tracking activities. Most of the experts pointed out that we had wrong assumptions regarding the activity of reviewing, resolving and prioritizing of issue reports. According to them, setting the priority should not be counted

as an issue-tracking activity, since it may be done by a manager or even a report submitter (which will not give any strong indication on the role of the activity initiator). Also, the reviewing activity should be detailed more carefully to distinguish it from the resolving activity (e.g., through checking the value of the assigned-to and resolution-status fields). Furthermore, a possible reviewing activity we didn't consider was if an unassigned issue report's resolution was set to duplicate and status to closed. Last but not least, we should consider the reopening of a closed issue report as a reviewing activity.

57% of the experts agreed that there are no real drawbacks of integrating *INExPERT* within issue-tracking practices. Other experts were concerned with having issues being "tossed" among unqualified assignees. Others highlighted a possible drawback as the lack of flexibility in being limited to the list of recommendations instead of manually choosing other unlisted assignees.

All seven experts agreed that using *INExPERT* would reduce the issue report's resolution time. Adding to *INExPERT*'s list the right information that relates the recommendations to project's sub-groups or package would help inexperienced or new team members select more suitable assignees using minimum efforts. Even if the selected assignee were not in charge of the fix, they would still be capable of reassigning the issue to the one in charge. This was a clear indication for us that the involved participants viewed *INExPERT* as a beneficial tool for speeding the workflow.

All seven experts stated that full automation of issue assignment would not be preferred since there is a lot of contextual information involved that requires human intervention within the assignment process. On the other hand, one expert suggested that full automation could be used to categorize newly submitted issue reports under sub-groups/packages, after which human intervention could start.

85% of the experts agreed that the issue-tracking activities performed by an assignee should influence their ranking within the recommendations list, since their expertise is associated with the amount and type of activities they perform on issue reports related to a specific category or topic. Additionally, one expert suggested that when we calculate the activity score of each assignee, we should consider how strongly this assignee is associated with the topic relevant to the newly added issue report .

The experts were asked to state the reasons behind having unassigned issue reports that were either resolved or closed. The most common causes for this phenomenon were (1) less amount of commitment and efforts from team members in keeping information up-to-date (since issue assignment is a voluntary act), (2) lack of experience, (3) lack of communication, and (4) the report was invalid or duplicate. The first two causes provided an indication that *INExPERT* could be beneficial in helping inexperienced and/or overloaded project members achieve a better quality of issue assignment.

4.3 EVALUATION

This section presents the evaluation of *INExPERT*'s accuracy and effectiveness, a process which followed a *summative evaluation approach* [71], which mainly focuses on examining a technology after it was introduced to assess whether its targeted goals were effectively achieved. To this end, our evaluation approach consisted of the analysis of

the experts' feedback on assessing the effectiveness of *INExPERT*'s recommendations, along with the benchmarking of its accuracy against an LDA-SVM–based approach (using several datasets belonging to different domains, thus ensuring diversity and better quality).

The goal of the summative evaluation of *INExPERT* was to gather a realistic impression on how satisfied our experts with the accuracy and effectiveness of *INExPERT*'s recommendations were within the targeted environment (i.e., scientific software projects). Additionally, we wanted to evaluate the value of *INExPERT*'s contribution given the standards of the current state-of-the-art assignee-recommendation approaches. To this end, we developed the formal prototype of *INExPERT* mentioned in Section 4.1.

4.3.1 *Experts' Feedback*

Acquiring feedback from the targeted community of scientific software on how effective *INExPERT*'s outcomes are requires a large amount of effort and time. A less cumbersome, more effective approach would be to work with domain experts who are well aware of the community, along with its formal and informal rules. Those can give us feedback about the targeted environment we want to deploy *INExPERT* in. To this end, we approached three domain experts within the ATLAS-Reconstruction project who helped us before during *INExPERT*'s formative evaluation (see Section4.2.1).

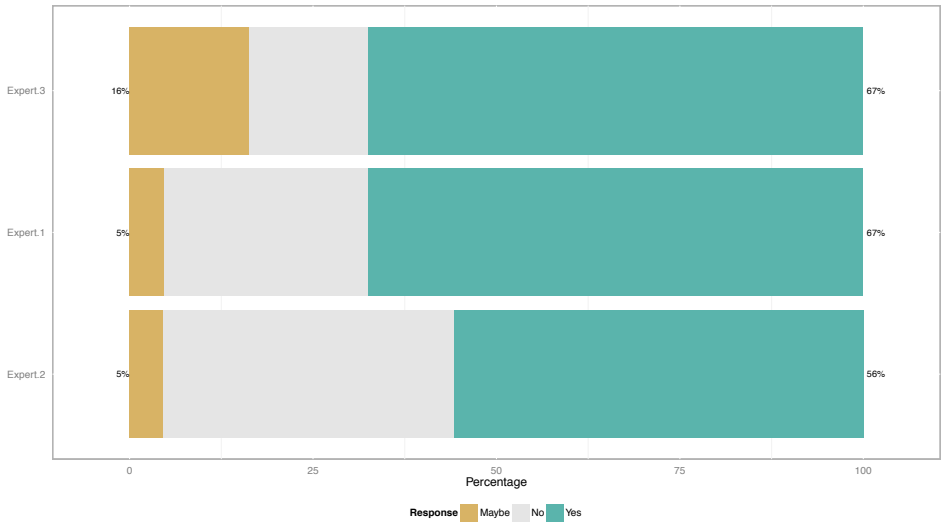

Figure 4.18: Experts' Judgement on Recommended Assignees Capabilities

We asked the three experts to evaluate 43 assignee recommendations for existing issue reports (dated between February 2011 and March 2012) generated by *INExPERT*'s formal prototype. In the evaluation, we used one criterion for determining the effectiveness of *INExPERT*'s assignee recommendations: **Assignee's Capability**. This refers to the needed expertise and correct role for the recommended assignees to fix or resolve a specific report (i.e., are the recommended assignees capable enough of fixing or reassigning this specific issue report?). To be able to reliably aggregate and compare

the evaluation metric from all participants' feedback, we asked them to choose from a simple level-of-agreement scale: No, Maybe, Yes.

For documentation purposes, we provided each participant with an evaluation form that consisted of (1) the details of 43 issue reports for identifying the needed expertise, (2) a ranked *INExPERT* list of assignee recommendations for each given issue report, and (3) participant's judgment (i.e., No/Maybe/Yes) and the justifications for choosing this specific level concerning the assignees capabilities within each list of recommendations. Figure 4.18 shows the experts' final judgments (i.e., No/Maybe/Yes) regarding the capability of recommended assignees in fixing or reassigning the 43 given issue reports. On average, all three experts stated that within 63.57% of the time, the assignees within the list of recommendations were either cable of fixing or reassigning the issue report. On the other hand, 36.44% of the time they were uncertain. To measure the reliability of the experts' feedback, we used a Fleiss' Kappa test; which indicated that experts judgments were fairly coherent: a fair level of agreement on both the issue reports rated with 'No' (kappa = 0.229 and p-value = 0.009) and 'Yes' (kappa = 0.398 and p-value = 0.000134).

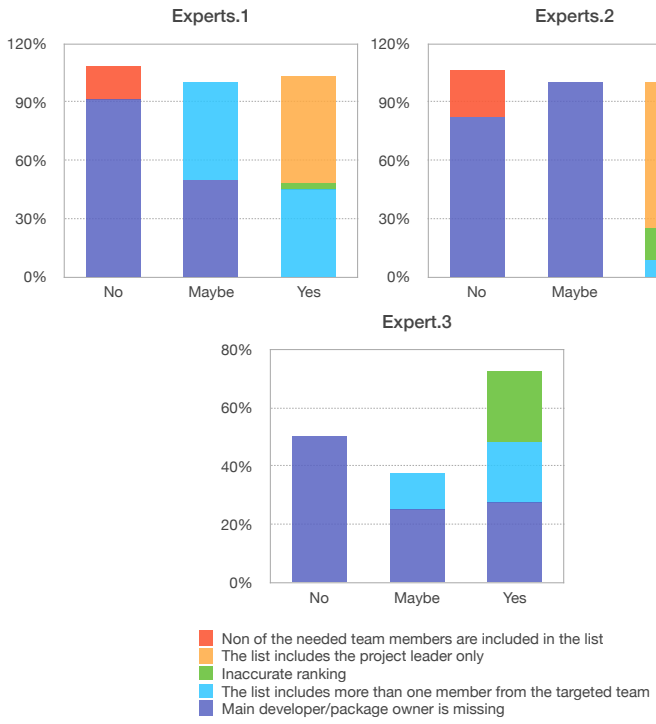

Figure 4.19: Experts' Judgements Vs. Justifcations

Figure 4.19 presents the justifications provided by all experts for each given judgment. The most common justification when judging with 'No' was that the list of recommended assignees didn't include the main developer/package owner. On the other hand, the most common justifications used when judging a 'Yes' were: "list included the project leader who won't fix it but can reassign it to a more suitable person", "list was ranked correctly" and "list included more than one member from the targeted team".

These outcomes imply that *INExPERT* was – in general – effective. Even when the experts stated that the main assignee was missing, the list usually included someone who is capable of reassigning the issue to a more suitable assignee. However, there definitely is a need for improvement within the areas of ranking and elimination; given *INExPERT*'s tendency to favor high level roles (e.g., project leaders) over other important roles (e.g., developers and package owners). To that end, more investigation on how to tune the activity score within the assignees' activity profiles is required for overcoming this issue.

4.3.2 *Benchmarking*

We selected a combined LDA/SVM technique [77] as the baseline reference to use for benchmarking *INExPERT*'s performance, since it represents the current state-of-the-art work in automated issue assignment. Using the probability matrix from the LDA computation as the feature vector for SVM classification results in (1) a smaller feature vector and (2) added stability when computing the SVM prediction results. SVM calculates a separating hyperplane between data points from different classes and tries to maximize the margin between them. For evaluating SVM against *INExPERT*, we used a multi-class SVM classifier, in order to predict a suitable class (i.e., assignee) from a given list of project members. The implementation of SVM was provided by Weka [28], a machine learning framework with a user interface (Explorer). We used it to get a prediction matrix that contains the probability distribution of having a certain issue report belonging to a certain class of assignees.

4.3.2.1 *Evaluation Dataset & Contextual Information gathering*

To evaluate *INExPERT*, we used issue-tracking repository data from three different software projects. The projects were selected to ensure diversity within the quality of issue reports, how they get assigned, and issue-tracking activities in general; so that we could draw unbiased interpretations. ATLAS-Reconstruction[6] is a mid-sized scientific software project. Eclipse BIRT[7] is a large open-source project with market orientation, and UNICASE[8] is a mid-sized software engineering research project.

The data obtained from these projects consisted of issue reports as well as the activity history (i.e., log files of user interactions within the issue-tracking repository). We transformed the database dumps from the three projects into a unified database. Unification made it possible to easily (1) filter unclassified and invalid reports, (2) obtain issue-tracking activities, and (3) extract LDA input features for the topic.

To gather contextual information regarding how issue reports within these investigated projects get assigned —which should help us understand the assignment of issue reports before applying *INExPERT* —we conducted a semi-structured interview that consisted of five open-answer questions. These investigated how issue reports are triaged (i.e., how issue reports are assessed to see if they describe a meaningful new problem or enhancement, how they are assigned to an appropriate developer, and how they are prioritized for further handling). The interview questions also focused

6 http://atlas.ch/
7 http://www.eclipse.org/birt/phoenix/
8 http://code.google.com/p/unicase/

on identifying who is responsible for assigning (or reviewing the assignment) of issue reports. We interviewed one to three senior developers within each investigated project, asking them the following questions:

1. Were you ever involved in the process of issue triaging? If yes, please describe your experience in details.
2. When an issue report is submitted, who gets to be notified? And How?
3. Who is responsible for triaging issue reports? Please describe the triaging process in details.
4. Who is responsible for assigning issue reports?
5. When a user assigns an issue report to a certain developer during submission, does anyone review his decision concerning the developer he chose? Or it's just reviewed by the developer himself directly?

Additionally, we analyzed the projects' documentation concerning the submission and assignment guidelines of issue reports published on the projects' wikis, in order to fill any gaps in our understanding of the interview responses.

In the ATLAS-Reconstruction project, any new issue report is usually assessed and assigned by the issue tracker owners (as they are the ones who get notified upon submission). However, other issue tracker members can take the initiative to assign and assess new issue reports, as this task is not restricted to the role of tracker owners (i.e., it is more of a voluntarily task). This is due to the busy schedule of the project members (since they are involved in many other research activities) so they have to collaborate on handling and resolving ingoing issues. Additionally, newly submitted issue reports can also be assigned by their reporters. However, if they are unable to determine a suitable developer, they usually assign it to one of the main managers or leave it unassigned. Consequently, new issue reports sometimes get to be closed without being assigned to a specific developer. This lack of tractability due to incomplete information makes it very difficult to verify or reassess the resolution of the incomplete issue reports.

The Eclipse BIRT project is divided into multiple components, each owned by a specific developer. When a new issue report is submitted, it is typically assigned to the component owner, who then assesses the issue report. If it is a valid issue, it is assigned to a suitable developer, who can then either accept or refuse it. This indicates that the task of assigning an issue is restricted to the component owner role. Therefore, only Eclipse committers and component owners can edit/move/reassign issue reports.

When an issue report is submitted to the UNICASE project, it is assigned by its reporter to a specific developer. However, if he is unable to determine a suitable developer, it is assigned to one of the two main committers. Either the assigned developer or the main committers assess the issue report and check if it is a valid issue, then work on it. In case the assignee is overloaded, or unable to resolve the issue, it is reassigned to another, more suitable developer. Table 4.4 gives a summary on the evaluation dataset and contextual information of the investigated projects.

Table 4.4: Overview of the Investigated Projects

Project	**ATLAS-Reco**	**BIRT**	**UNICASE**
Domain	Scientific Software	Market Orientation	Research
Issue Repository	Savannah	Bugzilla	UNICASE
Issue Reports are assigned by	Volunteer	Developer	Developer
Issue Assignment	Informal	Formal	Formal
Avg. Reports/day	0.73	3.39	0.28
Avg. Issue-Tracking Activities/day	2.34	11.34	1.60

4.3.2.2 *Training & Testing Dataset*

Not all issue reports were included in the evaluation dataset (i.e., training set and testing set, combined). Some issue reports were excluded to guarantee that the expertise of the assignee is determined based on a set of "completed" issue reports. To that end, we excluded all the reports that were not closed, assigned or resolved (i.e., issue reports having an undefined state). In addition, we adjusted the original assignee according to [1] for BIRT, since they state that if a report had a resolution of "fixed", label it with whoever changed the state of the report to "fixed". This is due to the fact that sometimes the given email for the assignee in the repository is not of an individual person, but rather of a group that is responsible for a certain project component.

The criteria for selecting the evaluation dataset were based on the time period over which the issue reports had occurred, as suggested by [1]. Furthermore, the dataset was split into a training dataset and a testing dataset. Table 4.5 gives an overview of the total number of issue reports, project members and time frame included in our training and testing datasets.

Table 4.5: Overview of Training and Testing Datasets

Training Set			
Project	ATLAS	BIRT	UNICASE
#Reports	149	506	98
#Project Members	47	62	12
Time Period	7 months	8 months	5 months
Testing Set			
Project	ATLAS	BIRT	UNICASE
#Reports	46	118	28
#Project Members	22	37	9
Time Period	1 month	1 month	1 month

Selecting the training set was based on including the most recent issue reports in the project. The duration we choose varied from one project to another, depending on the quantity needed to enable us extract topics and user activities from the issue tracking activities logs. Table 4.6 provides an overview of the issue-tracking activities distribution within the training set. Consequently, the testing set was chosen over a time frame of one month after the training set's time frame, ensuring it is consistent and up-to-date with the training set.

After pre-processing the dataset as mentioned in section 4.1.1, we extracted a 10-topics $\times$ 10-terms model for the issue reports of both the training and the testing datasets. We added a constraint for each issue report to be associated with only 10 topics, and for each topic to be associated with only 10 terms. However, a term can be associated with more than one topic.

Table 4.6: Overview of Issue-Tracking Activities Distribution in Training Set

% of Issue-Tracking Activities			
	ATLAS	BIRT	UNICASE
#Assigning	35%	32%	15%
#Resolving	32%	31%	36%
#Reviewing	33%	37%	49%
Weight of Issue-Tracking Activities			
	ATLAS	BIRT	UNICASE
Assigning Weight	3.50	3.21	1.48
Resolving Weight	3.16	3.09	3.62
Reviewing Weight	3.35	3.70	4.91

4.3.2.3 *Benchmarking Results*

To evaluate how our approach performs in comparison with the existing LDA-SVM–based approach, we evaluated data from three different projects. A comparison was done over (1) finding the actual assignee by measuring the actual assignee hit ratio, and (2) evaluating the ranking by measuring the top-n ranking hit ratio.

Actual Assignee Hit Ratio: We compared the two approaches based on hit ratio (i.e., having the actual assignee within the list of recommended assignees, see Figure 4.20).

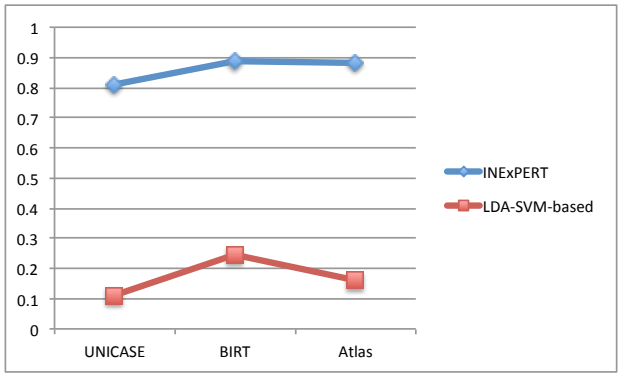

Figure 4.20: Hit ratio for having the main resolver against the total number of issues

INExPERT counts a "hit" when the list of recommended assignee contains the actual assignee of the new issue report. For LDA-SVM, a "hit" is considered if it predicts the actual assignee correctly. Being a binary classifier, SVM does not perform better than *INExPERT*, which produces a ranked list of assignees for each issue report. As the results show, LDA-SVM performed better in the BIRT project, but the results are still not close to our approach.

Top-n Hit: We did a Top-n hit comparison between the two approaches. For LDA-SVM, we used the probability distribution for each of the candidate assignees as their ranking (in decreasing order), so as to have a ranked list for both approaches. The hit ratio in top 1, top 3, top 5, and top 10 was computed for each project. We considered a hit for *INExPERT* if the ranked list contained any assignee who has performed either of the three activities (assigning, reviewing, or resolving). Performing any of these activities implies that the assignee has some expertise in the issue report's topic.

Figure 4.21 shows the hit ratio for the Atlas-Reconstruction project using activity and LDA-SVM approaches. Over a testing set of 40 issue reports within the project,

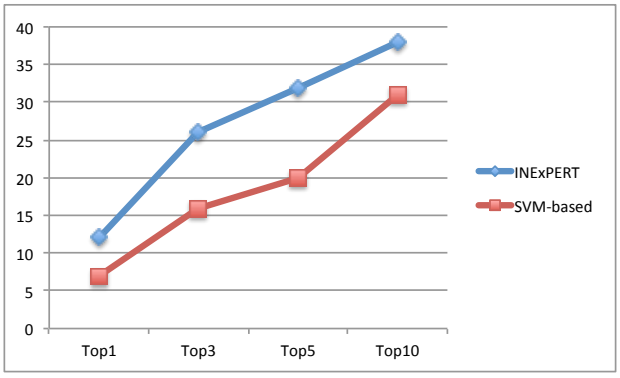

Figure 4.21: Top-n Hits, INExPERT vs LDA-SVM; Atlas

INExPERT performed better in comparison with LDA-SVM for top 1, top 3, top 5, and top 10 hits. Considering the small testing datasets, our approach managed to correctly predict the assignee for almost all the issue reports. In contrast, the assignee predicted by LDA-SVM for three-quarters of the issue reports were within the top 10 hits.

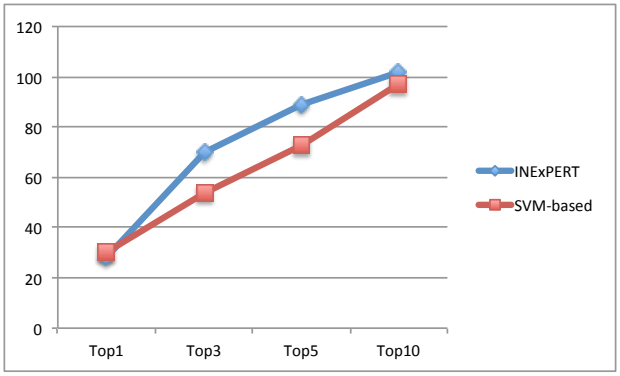

Figure 4.22: Top-n Hits, INExPERT vs LDA-SVM; Birt

BIRT (see Figure 4.22) had three times as many issue reports as Atlas-Reconstruction had in its testing set. LDA-SVM performed almost the same as *INExPERT* in this project for top 1 hit, but our approach excelled in top 3, top 5, and top 10 hits. We believe this is because *INExPERT* only considers the most relevant assignee during the candidate selection phase, so that the number of assignees in the final list is eventually smaller in comparison with LDA-SVM. We also observed that LDA-SVM falls behind in the top 3 and top 5 tests, but eventually manages to trail our approach in the top 10 test (with linear progression).

For the UNICASE project (see Figure 4.23), both approaches gave almost similar results. Although *INExPERT* performed slightly better for top 1 and top 3 hits, it fell behind the LDA-SVM approach in the rest. A common observation in all three projects was that *INExPERT* performs quite well for top 1, top 3, and top 5 hits; but LDA-SVM manages to reach better results at the end. This is because *INExPERT* recommends the suitable assignee for most of the issue reports within top 5 hits only; while LDA-SVM with linear progression —as shown for all projects —has the suitable assignees spread across different hit ratios. The reason for this is that *INExPERT* uses elimination for creating a list of candidates, while LDA-SVM considers all project members against

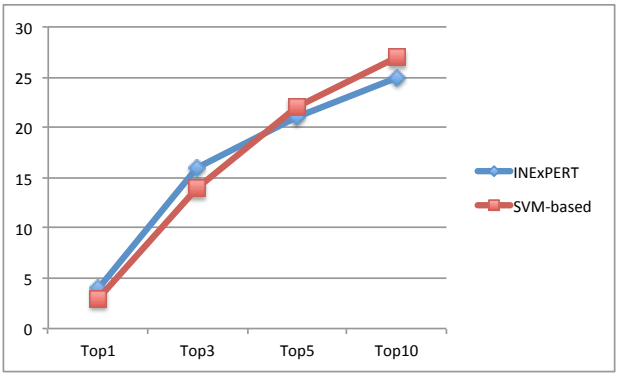

Figure 4.23: Top-n Hits, INExPERT vs LDA-SVM; UNICASE

each issue report. Since UNICASE consists of rather a small dataset, the elimination performed by *INExPERT* came in favor of the LDA-SVM results, especially at top 10 hit ratio, where the whole nine project members were included in the LDA-SVM probability distribution list (see Figure 4.23).

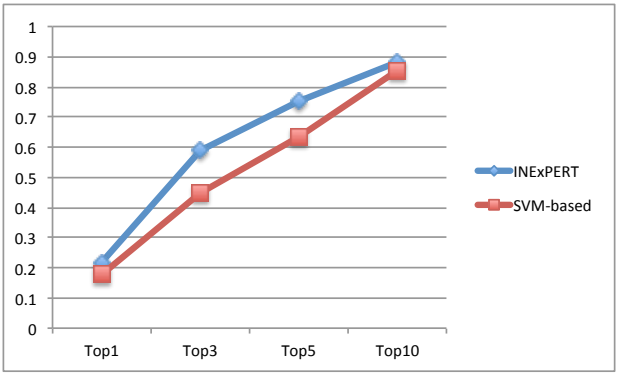

Figure 4.24: Overall hit ratio

This further becomes obvious looking at the overall hit ratio calculated for all three projects (see Figure 4.24), where *INExPERT* achieves a hit ratio of 88%, while LDA-SVM is close enough by 85% for top 10 hits. Hence, we believe our approach can be quite useful to predict a suitable assignee, if the recommendation list size has to be kept shorter with better precision.

5 CONCLUSION AND FUTURE WORK

This chapter highlights the outcomes of our research work, as well as details of the limitations within the conducted empirical study and the issue assignees recommendation technique *INExPERT*. We also conclude our work with remarks related to future directions for research.

5.1 RESEARCH OUTCOME HIGHLIGHTS

We presented the results of an empirical study which investigates issue tracking activities in software engineering and scientific software projects. The study analyzed issue reports and issue history logs, as well as answers given by 612 project participants. The results of the study can help software engineers define the necessary improvements of issue tracking practices in both software engineering and scientific software projects.

Additionally, we applied LDA-SVM as well as an issue tracking activity-based technique, '*INExPERT*' for issue assignees recommendation. Both techniques were evaluated using bug reports from three existing projects: (1) ATLAS-Reconstruction is a mid-sized scientific software project, (2) Eclipse BIRT is a large open-source project with market orientation, and (3) UNICASE is a mid-sized software engineering research project. This work confirms the results from previous research work that SVM is an efficient solution for the classification task in the case that the probability distribution of predicted classes is considered for ranking the developers in the candidate list. *INExPERT* performs better in most of the projects, albeit requiring the mining of an activity profile for each of the developers. As seen from the results, *INExPERT* performs significantly better if the recommendation list size is a constraint, i.e., the number of produced recommendations should be relatively small. Additionally, it is able to provide consistent results across varying dataset sizes. This is due to the stricter selection criteria of topic associations used for the new issue report on which the selection of assignees in the candidate list is based. Hence, the most relevant developers are the only ones to make it to the final ranked recommendation list.

We believe that combining both approaches can be an interesting option as well. This would facilitate having a recommended list introducing the intersection of the results of both approaches. In addition, an experiment with senior developers and managers in a large-scale scientific software project confirms the precision of *INExPERT* and its suitability for the scientific software domain. The outcomes imply that *INExPERT* was in general effective. Even when the experts stated that the main assignee was missing, the list usually included someone who is capable of reassigning the issue to a more suitable assignee.

5.2 RESEARCH LIMITATIONS

The generalizability of the empirical study we conducted to the scientific software as well as software engineering domain is limited since we only studied single projects. However, the selected projects were large and included a total of 612 participants. Therefore, the results for the single projects are representative. Another threat to validity is the survey: participants contributed on a voluntary basis and this self-selection can introduce unwanted bias.

Regarding *INExPERT* performance, there is a need for improvement within the areas of ranking and elimination given *INExPERT*'s tendency to favor high-level roles over other important roles.

5.3 FUTURE WORK

While the study detected differences in issue tracking activities in software engineering and scientific projects, further research is needed to detect the possible causes for these differences. Furthermore, we can reuse the study metrics and survey questions as an assessment framework for evaluating the quality of issue tracking practices in different projects. However, a suitable benchmark is needed to indicate what is a positive and a negative evaluation result. To that end, more interviews and observational studies of participants and their issue tracking activities should be conducted. Additionally, we aim at applying and evaluating our assignee recommendation technique *INExPERT* on an even larger dataset, e.g., Eclipse, Firefox. Moreover, we plan to do an evaluation over time analysis, as opposed to the state-based (snapshot of the bug repository) evaluation done in this work. We further plan to investigate how to tune the activity score within the assignees' activity profiles, in order to overcome the tendency to favor high-level roles. In addition, we plan to investigate the use of the activity profiles within *INExPERT* as an add-on to enhance the performance of existing assignee recommendation techniques.

5.4 FINAL THOUGHTS

We support the strategy suggested by Pawlik et al. [61] and Lethbridge et al. [44], which states that rather than forcing members of scientific software projects to be committed to formal and strict guidelines within software engineering practices, software engineers should conduct more research to measure how these practices are perceived and used, in order to identify what exactly is needed to make them more useful for the scientific software domain.

Part I

APPENDIX

A

APPENDIX

A.1 LAYOUT OF THE INTERVIEW USED IN INEXPERT FORMATIVE EVALUATION

1. What do you think is the reason behind having closed issue reports that are unassigned?
2. In the first stage of the experiment within issue report #1, what were the main aspects you considered when choosing the assignee?
3. In your opinion what is the impact of using INEXPERT on issue reports resolution time?
   - Increase
   - Decrease

   Please state the reason behind your answer.
4. In the second stage of the experiment with issue report #1, what were the main aspects you consider when choosing the assignee from the list of recommendation?
5. In your opinion, what would be the main drawback of integrating INEXPERT within your issue tracker?
6. Would you have preferred if the issue assignment was done automatically without human intervention?
7. Are the following descriptions of issue tracking activities correct or not? (Answer with Yes or No) Please state the reason behind your answer.
   - Resolving: change in resolution field to fixed, won't fix, need info, unproducible.
   - Managing–assigning activity: Change in assigned to field.
   - Managing–reviewing activity:
     - change in closed field, that is not done by same person who changed the resolution.
     - change in resolution invalid or duplicate.
8. Do you agree that the issue tracking activities of a project member influences his/her ranking within the list of assignee recommendations? (Strongly agree – Strongly disagree)

A.2 A COPY OF THE SURVEY USED IN OUR COMPARATIVE STUDY

Bug Tracking Practices

I. About Your Project

Gathers information about the project you are currently working on.

1. Which project are you contributing to?

Please choose *only one* of the following:
o ATLAS-Reconstruction
o Belle(2)
o CMS
o Eclipse
o UNICASE
o Other: ___

2. Which of the following best describes your main role within the project?

Please choose *only one* of the following:
o Release or project manager
o Developer
o Developer who is also a user of the software
o User who is not involved in the development of the software
o Tester
o Documentation maintainer
o Other: _

3. How long have you been enrolled within your project?

Please choose *only one* of the following:
o Less than 1 year
o 1-2 years
o 2-5 years
o More than 5 years

II. Bug Tracking Practices

Determines how bug tracking is being practiced within your project

1. How would you rate the bug tracking practices within your project?

Please choose the appropriate response for each item:

	-2 -1 0 1 2
Ambiguous | Clear	o o o o o
Obsolete | Up to date	o o o o o
Disintegrated from development cycle | integrated within the development cycle	o o o o o
Verbal | Documented	o o o o o

2. Which of the following activities do you mostly do?

Please choose *all* that apply:
o Submitting a bug report
o Reviewing & assigning a bug report
o Recording the procedure of handling & resolving a bug report
o Discussing a bug report with team members
o Prioritizing & scheduling a bug report
o Resolving a bug report
o Other: ___

3. What was the biggest benefit you've gained from using a bug tracking system within your project?

Please choose *only one* of the following:
o Sharing of bug reports information within one place
o Communication with developers
o monitoring of development progress
o Increased awareness and responsiveness to issues
o Easier release planning
o None
o Other: ___

4. In your project, how often do you use a bug tracking system to fulfill that following activities?
Rarely ≈ yearly / Sometimes ≈ monthly / Often ≈ weekly / Usually ≈ daily

Please choose the appropriate response for each item:
Submitting a bug report
Reviewing & assigning a bug resort
Recording the procedure of handling & resolving a bug report
Discussing a bug report with team members
Prioritizing & scheduling a bug report

5. Which of the following bug tracking methods have you used before?

Please choose *all* that apply:
o Bug tracking system
o Comments in source code
o Discussion Forum or mailing list
o Private emails
o Phone and face-to-face communication
o Wiki
o Shared files (e.g. excel sheets)
Other: ___

6. Which of the following tools would you prefer to use when performing the following activities?

Please choose the appropriate response for each item:
Submitting a bug report
Reviewing & assigning a bug report
Recording the procedure of handling & resolving a bug report
Discussing a bug report with team members
Prioritizing & scheduling a bug report
Resolving a bug report

7. What is the average time, you activity spend on completing the following activities?

Please choose the appropriate responses for each item:
Submitting a bug report
Reviewing & assigning a bug report
Recording the procedure of handling & resolving a bug report
Discussing a bug report with team members
Prioritizing & scheduling a bug report
Resolving a bug report

III. Problems and Enhancements

Evaluates the problems your team encounters and your suggestions on how to solve them.

1. How often do you encounter each of the following problems? Do you agree that these problems cause severe overhead for your team?
Rarely ≈ yearly / Sometimes ≈ monthly / Often ≈ weekly / Usually ≈ daily

Please choose the appropriate response for each item:
Assigning a bug report to the incorrect developer
Assigning duplicate bug reports to different developers
Not recording the procedure of handling & resolving a bug report
Not setting a priority or properly scheduling a bug report
Not tracking a bug that was fixed on the fly
Submitting a bug report with an incorrect category
Using different bug tracking methods within the same team

2. The following activities produce an overhead that has a negative impact on the overall project performance, Do you agree or disagree?

Please choose the appropriate response for each item:
Submitting a bug report
Assigning a bug to the correct developer or category
Detecting & linking duplicated bugs
Recoding the procedure of handling & resolving a bug report
Discussing a bug report among team members
Prioritizing & scheduling a bug report
Maintaining the bug tracking role or system

3. Can you recall any other problems or overheads that weren't previously mentioned?
Please write your answer here:

4. How would you solve some of the previously mentioned problems and overheads?
Please write your answer here:

IV. Context

The questions within this section will help us gather information about your educational and professional expertise.

1. What is your main educational background?

Please choose *all* that apply:
o Computing
o Mathematics
o Physics
o Software engineering
Other: ___

2. How do you rate your level of software engineering knowledge (i.e. using and adopting state-of-the-art software engineering tools and techniques)?

Please choose *only one* of the following:
o Very experienced
o Experienced
o Somewhat experienced
o Unexperienced
o Very inexperienced

Thank you

As a token of our appreciation on your participation, we would like to have your email address in order for you to take part in a lottery, where you can win a 100 euro Amazon gift card or some other interesting girts.
Please write your answer(s) here:
Email: : ___
Survey Result If you would like to get a summary of the survey results without providing us your email address, please check here in about 4 weeks from now.

Submit your survey.

Thank you for completing this survey.

BIBLIOGRAPHY

[1] John Anvik and Gail C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.*, 20:10:1–10:35, 2011.

[2] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 361–370. ACM, 2006.

[3] A. April and A. Abran. *Software maintenance management: evaluation and continuous improvement*. Wiley-IEEE Computer Society Press, 2012.

[4] A Bachmann and A Bernstein. When process data quality affects the number of bugs: Correlations in software engineering datasets. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 62–71, May 2010.

[5] Adrian Bachmann and Abraham Bernstein. Software process data quality and characteristics: a historical view on open and closed source projects. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, pages 119–128. ACM, 2009.

[6] V R Basili, J C Carver, D Cruzes, L M Hochstein, J K Hollingsworth, F Shull, and M V Zelkowitz. Understanding the High-Performance-Computing Community: A Software Engineer's Perspective, 2008.

[7] Victor R Basili, Forrest Shull, and Filippo Lanubile. Building knowledge through families of experiments. *Software Engineering, IEEE Transactions on*, 25(4):456–473, 1999.

[8] O. Baysal, M.W. Godfrey, and R. Cohen. A bug you like: A framework for automated assignment of bugs. In *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, pages 297 –298, may 2009.

[9] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 308–318. ACM, 2008.

[10] Ronald F Boisvert. Mathematical software: past, present, and future. *Mathematics and computers in simulation*, 54(4):227–241, 2000.

[11] Barrett R. Bryant, Jeff Gray, and Marjan Mernik. Domain-specific software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 65–68. ACM, 2010.

[12] Jeffrey Carver. Empirical studies in end-user software engineering and viewing scientific programmers as end-users. In *proceedings of Dagstuhl Seminar on End-User Software Engineering, Internationales Begegnungs-und Forschungszentrum für Informatik (IBFI) Schloss Dagstuhl*. Citeseer, 2007.

[13] Jeffrey Carver, Dustin Heaton, Lorin Hochstein, and Roscoe Bartlett. Self-perceptions about software engineering: A survey of scientists and engineers. *Computing in Science & Engineering*, 15(1):7–11, 2013.

[14] Jeffrey C Carver. Software engineering for computational science and engineering. *Computing in Science & Engineering*, 14(2):8–11, 2012.

[15] Jeffrey C Carver, L Hochstein, Richard P Kendall, Taiga Nakamura, Marvin V Zelkowitz, Victor R Basili, and Douglass E Post. Observations about software development for high end computing. *CTWatch Quarterly*, 2(4A):33–37, 2006.

[16] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. Software development environments for scientific and engineering software: A series of case studies. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society.

[17] Yguaratã Cerqueira Cavalcanti, Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Towards understanding software change request assignment: a survey with practitioners. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pages 195–206. ACM, 2013.

[18] D. Čubranić and Gail C. Murphy. Automatic bug triage using text categorization. In *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.

[19] Don A Dillman, Robert D Tortora, and Dennis Bowker. Principles for constructing web surveys. *Joint Meetings of the American Statistical Association*, 1998.

[20] Kai-Bo Duan and S Sathiya Keerthi. Which is the best multiclass svm method? an empirical study. In *Multiple Classifier Systems*, pages 278–285. Springer, 2005.

[21] Roshanak Farhoodi, Vahid Garousi, Dietmar Pfahl, and Jonathan Sillito. Development of scientific software: A systematic mapping, a bibliometrics study, and a paper repository. *International Journal of Software Engineering and Knowledge Engineering*, 23(04):463–506, 2013.

[22] Stuart Faulk, Eugene Loh, Michael L Van De Vanter, Susan Squires, and Lawrence G Votta. Scientific computing's productivity gridlock: How software engineering can help. *Computing in science & engineering*, 11(6):30–39, 2009.

[23] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. The measurement of interrater agreement. *Statistical methods for rates and proportions*, 2:212–236, 1981.

[24] Thomas Fritz, Gail C. Murphy, and Emily Hill. Does a programmer's activity indicate knowledge of code? In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 341–350. ACM, 2007.

[25] Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. Measuring developer contribution from software repository data. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 129–132. ACM, 2008.

[26] Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(Suppl 1):5228–5235, 2004.

[27] S.R. Gunn. Support vector machines for classification and regression. Technical report, Dept. of Electronics and Computer Science, University of Southampton, 1998. Address: Southampton, U.K.

[28] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, 2009.

[29] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8. IEEE Computer Society, 2009.

[30] D Heaton, JC Carver, R Bartlett, K Oakes, and L Hochstein. The Relationship between Development Problems and Use of Software Engineering Practices in Computational Science & Engineering: A Survey. In *Proceedings of the E-Science, IEEE 8th International Conference*, 2012.

[31] Jonas Helming, Holger Arndt, Zardosht Hodaie, Maximilian Koegel, and Nitesh Narayan. Automatic assignment of work items. In *Evaluation of Novel Approaches to Software Engineering*, volume 230 of *Communications in Computer and Information Science*, pages 236–250. Springer Berlin Heidelberg, 2011.

[32] Michael A Heroux and James M Willenbring. Barely sufficient software engineering: 10 practices to improve your cse software. In *Software Engineering for Computational Science and Engineering, 2009. SECSE'09. ICSE Workshop on*, pages 15–21. IEEE, 2009.

[33] Kim Herzig and Andreas Zeller. Mining bug data. In *Recommendation Systems in Software Engineering*, pages 131–171. Springer, 2014.

[34] MD Hoffman, DM Blei, and Francis Bach. Online learning for latent dirichlet allocation. *Advances in Neural Information Processing Systems*, 23:856–864, 2010.

[35] Kamal Hossen, Huzefa H Kagdi, and Denys Poshyvanyk. Amalgamating source code authors, maintainers, and change proneness to triage change requests. In *ICPC*, pages 130–141, 2014.

[36] James Howison and James D Herbsleb. Scientific software production: incentives and collaboration. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 513–522. ACM, 2011.

[37] Akinori Ihara, Masao Ohira, and Ken-ichi Matsumoto. An analysis method for improving a bug modification process in open source software development. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops - IWPSE-Evol '09*, page 135. ACM Press, August 2009.

[38] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 111–120. ACM, 2009.

[39] D.F. Kelly. A Software Chasm: Software Engineering and Scientific Computing. *IEEE Software*, 24(6), 2007.

[40] Diane Kelly. Industrial scientific software: A set of interviews on software development. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 299–310, Riverton, NJ, USA, 2013. IBM Corp.

[41] Diane Kelly, Robert Gray, and Yizhen Shao. Examining random and designed tests to detect code mistakes in scientific software. *Journal of Computational Science*, 2(1):47–56, 2011.

[42] Sarah Killcoyne and John Boyle. Managing chaos: lessons learned developing software in the life sciences. *Computing in science & engineering*, 11(6):20–29, 2009.

[43] Taeksu Kim, Chanjin Park, and Chisu Wu. Mock object models for test driven development. In *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*, pages 221–228, Aug 2006.

[44] Timothy C Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *Software, IEEE*, 20(6):35–39, 2003.

[45] Jingyue Li, Tor Stalhane, Reidar Conradi, and JMW Kristiansen. Enhancing Defect Tracking Systems to Facilitate Software Quality Improvement. *Software, IEEE*, 29(2):59–66, March 2012.

[46] Yang Li. Reengineering a scientific software and lessons learned. In *Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering*, pages 41–45. ACM, 2011.

[47] Douglass E li2011reengineering and Richard P Kendall. Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulations: Lessons learned from asci. *International Journal of High Performance Computing Applications*, 18(4):399–416, 2004.

[48] Rafael Lotufo, Leonardo Passos, and Krzysztof Czarnecki. Towards improving bug tracking systems with game mechanisms. *2012 9th IEEE Working Conference on Mining Software Repositories MSR*, pages 2–11, 2012.

[49] Erika S. Mesh and J. Scott Hawker. Scientific software process improvement decisions: A proposed research strategy. In *Proc. SE-CSE*, pages 32–39. IEEE, 2013.

[50] Greg Miller. A scientist's nightmare: Software problem leads to five retractions. *Science*, 314(5807):1856–1857, 2006.

[51] Audris Mockus and James D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 503–512. ACM, 2002.

[52] J. Yates Monteith, John D. McGregor, and John E. Ingram. Scientific research software ecosystems. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW '14, pages 9:1–9:6. ACM, 2014.

[53] Hoda Naguib and Yang Li. (position paper) applying software engineering methods and tools to cse research projects. *Procedia Computer Science*, 1(1):1505–1509, 2010.

[54] Hoda Naguib, Nitesh Narayan, Bernd Brugge, and Dina Helal. Bug report assignee recommendation using activity profiles. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 22–30. IEEE, 2013.

[55] Naresh Kumar Nagwani and Shrish Verma. Rank-me: A java tool for ranking team members in software bug repositories. *Journal of Software Engineering and Applications*, 5(4):255–261, 2012.

[56] Robert G Newcombe. Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in medicine*, 17(8):873–890, 1998.

[57] Tung Thanh Nguyen, Anh Tuan Nguyen, and Tien N Nguyen. Topic-based, time-aware bug assignment. *ACM SIGSOFT Software Engineering Notes*, 39(1):1–4, 2014.

[58] L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10. ACM, 2010.

[59] Jin-Woo Park, Mu-Woong Lee, Jinhan Kim, Seung won Hwang, and Sunghun Kim. Costriage: A cost-aware triage algorithm for bug reporting systems. In *AAAI*, 2011.

[60] J.W. Paulson, Giancarlo Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30 (4):246–256, 2004.

[61] Aleksandra Pawlik, Judith Segal, and Marian Petre. Documentation practices in scientific software development. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on*, pages 113–119. IEEE, 2012.

[62] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.

[63] Prakash Prabhu, Thomas B Jablin, Arun Raman, Yun Zhang, Jialu Huang, Hanjun Kim, Nick P Johnson, Feng Liu, Soumyadeep Ghosh, Stephen Beard, et al. A survey of the practice of computational science. In *State of the Practice Reports*, page 19. ACM, 2011.

[64] Md. Mainur Rahman, Guenther Ruhe, and Thomas Zimmermann. Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, pages 439–442, Washington, DC, USA, 2009. IEEE Computer Society.

[65] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12: 23–49, 1999.

[66] G. Robles. Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 171 –180, may 2010.

[67] Ripon K Saha, Sarfraz Khurshid, and Dewayne E Perry. An empirical study of long lived bugs. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 144–153. IEEE, 2014.

[68] Neil J Salkind. *Encyclopedia of measurement and statistics*, volume 1. Sage Publications, Inc, 2007.

[69] Rebecca Sanders and Diane Kelly. Dealing with risk in scientific software development. *IEEE software*, 25(4):21–28, 2008.

[70] David Schuler and Thomas Zimmermann. Mining usage expertise from version archives. In *Proceedings of the 2008 international working conference on Mining software repositories*, MSR '08, pages 121–124. ACM, 2008.

[71] Michael Scriven. Beyond formative and summative evaluation. 1991.

[72] Judith Segal. Some problems of professional end user developers. In *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on*, pages 111–118. IEEE, 2007.

[73] Judith Segal. Software development cultures and cooperation problems: A field study of the early stages of development of software for a scientific community. *Computer Supported Cooperative Work (CSCW)*, 18(5-6):581–606, September 2009.

[74] Judith A Segal and Chris Morris. Developing software for a scientific community: some challenges and solutions. 2011.

[75] Ramin Shokripour, John Anvik, Zarinah M. Kasirun, and Sima Zamani. Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation. In *IEEE International Working Conference on Mining Software Repositories*, pages 2–11, 2013.

[76] J. Singer. Practices of software maintenance. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 139 –145, nov 1998.

[77] Kalyanasundaram Somasundaram and Gail C Murphy. Automatic categorization of bug reports using latent dirichlet allocation. *Proceedings of the 5th India Software Engineering Conference*, pages 125–130, 2012.

[78] DE Stevenson. A critical look at quality in large-scale simulations. *Computing in Science and Engineering*, 1(3):53–63, 1999.

[79] M. Steyvers and T. Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

[80] Greg Wilson, DA Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, et al. Best practices for scientific computing. *PLoS biology*, 12(1): e1001745, 2014.

[81] Wenjin Wu, Wen Zhang, Ye Yang, and Qing Wang. Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*, pages 389 –396, dec. 2011.

[82] Xihao Xie, Wen Zhang, Ye Yang, and Qing Wang. Dretom: developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, PROMISE '12, pages 19–28. ACM, 2012.

[83] Geunseok Yang, Tao Zhang, and Byungjeong Lee. Utilizing a multi-developer network-based developer recommendation algorithm to fix bugs effectively. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1134–1139. ACM, 2014.

[84] Liu Yingbo, Wang Jianmin, and Sun Jiaguang. A machine learning approach to semi-automating workflow staff assignment. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, pages 340–345. ACM, 2007.

[85] L. Yu and K. Chen. Evaluating the post-delivery fault reporting and correction process in closed-source and open-source software. In *Software Quality, 2007. WoSQ'07: ICSE Workshops 2007. Fifth International Workshop on*, page 8. IEEE, 2007.

[86] Yury V Zaytsev and Abigail Morrison. Increasing quality and managing complexity in neuroinformatics software development with continuous integration. *Frontiers in neuroinformatics*, 6, 2012.

[87] Tao Zhang and Byungjeong Lee. An automated bug triage approach: A concept profile and social network based developer recommendation. In *Intelligent Computing Technology*, volume 7389, pages 505–512. Springer Berlin Heidelberg, 2012.

[88] Donald W Zimmerman. Teacher's corner: A note on interpretation of the paired-samples t test. *Journal of Educational and Behavioral Statistics*, 22(3):349–360, 1997.

[89] Thomas Zimmermann, Rahul Premraj, Jonathan Sillito, and Silvia Breu. Improving bug tracking systems. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 247–250. IEEE, 2009.