UNIVERSITY OF CALIFORNIA,
IRVINE

Expertise Location Approaches and Systems in Software Engineering

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Software Engineering

by

Zhendong Wang

Thesis Committee:
Professor David Redmiles, Chair
Associate Professor James Jones
Assistant Professor Yi Wang

2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my parents who supported my education overseas.

I would like to thank professor Dr. David Redmiles who adopted me into his academic family, and professor Dr. Anita Sarma who led me to the world of Software research.

ABSTRACT OF THE THESIS

Expertise Location Approaches and Systems in Software Engineering

By

Zhendong Wang

Master of Science in Software Engineering

University of California, Irvine, 2018

Professor David Redmiles, Chair

Successful software engineering activities require qualified software developers with proper expertise. Although expertise has been studied for many years and various expertise location approaches have been postulated, new approaches and opportunities are emerging today because of the rise of code hosting and knowledge sharing sites. As a step towards understanding the past work and the present opportunities in the context of today's software engineering practice, we perform a systematic literature survey. In analyzing the literature, we identify two broad categories of expertise research: 1) identifying the characteristics of experts, and 2) locating experts. The studies in the latter category can be further classified into three subcategories, which are: i) locating expertise by leveraging the organizational setting; ii) by mining historical artifacts; and iii) by knowledge sharing. Our analysis also identifies the major limitations of existing work, including a disconnect between early expertise studies and current location approaches; an over reliance on the experience-based model to measure expertise; a neglect on the constraints for coordination; and a lack of empirical evaluation in the real-world context, among others. Finally, we highlight research trends and promising directions for future research.

Chapter 1

Introduction

Thriving with automation, the fields of artificial intelligence and machine learning techniques have been fundamentally changing software and traditional industries. The advance of these fields is transforming the way that we work and live. Machines and algorithms replace human labor in automated routine tasks, but in other cases, automation is amplifying human labor, including in Software Engineering.

Since Software Engineering is a human centered activity [33], effectively managing human resource may significantly enhance the project productivity and collaboration quality [15]. Moreover, successful software engineering activities require qualified developers with proper expertise to complete the task efficiently with higher than average performance.

A crucial aspect of managing human resource is locating expertise. Experts are able to recall their previous experience in similar work [11, 30] with outstanding information process ability [77]. Various tasks in professional software development, such as employees training and hiring, locating the best person to perform product maintenance, and collaboration among teammates, all may benefit from effective expertise location [10]. However, there is still a lack of general understanding on expertise location approaches and techniques from a systematic

review perspective. To further investigate this issue, we also intend to explore how to evaluate the effectiveness of these location techniques in software engineering practices. To address the above practical problem, this survey study reviews related literature on expertise location approaches and systems within the field of Software Engineering and Computer Supported Collaborative Work.

We take a systematic review strategy for this literature survey study. We compile a literature repository of 120 studies by querying four major online digital libraries (ACM Digital Library, Elsevier ScienceDirect, IEEE Xplore, Springer Link). We perform our literature review analysis through a six-step process, including steps of citation searching (snowball sampling), and manual literature addition based on expert suggestions, and so on. After applying our review protocol and examining studies within our literature repository, 48 are identified as primary based on their research focuses (corresponding to the research question of this study) and impact to the field.

By summarizing a view of the developing history of expertise location approaches and systems, we found that the granularity of the expertise location approaches has been becoming smaller, which are from *performing a general job* to *finishing a specific task*. According to the analysis on our literature repository, we identified two board categories of expertise research. The first category expertise research identifies the characteristics of expertise [30, 58, 78]. These studies conducted field observation [56] to explore how experts communicate and solve problems in the real-world context. Further other studies employed lab experiments to empirically compare the difference between expert and novice based on their behavior while completing programming tasks [10, 58]. The second category aims to locate expertise through various approaches. This category can be further classified into three subcategories, locating expertise by leveraging the organizational setting manually [90]; mining historical artifacts [7, 35, 60, 72, 74]; and analyzing the social network and the content for knowledge sharing [50, 61]. As the automated location techniques developing, and the emergence of

new collaborative models such as the pull request model in open source community, more specific types of expertise have been identified for open source collaboration [16, 22, 93]. We found recent studies have shifted their focuses mostly to mining historical artifacts. Also, as the knowledge sharing platforms emerging particularly for software engineering, there is a lack of observational study to determine the role of knowledge sharing sites in expertise location activities.

Our review also identifies the major limitations of existing research. First, there is a disconnect between early expertise studies and current location approaches. Our analysis suggests that mining historical artifacts is common practice in Software Engineering, and current automated location studies tend to focus mining historical data of expert's activity and artifacts which indicates their experience [22, 74, 93]. However, early cognitive studies identify the expertise based on monitoring the performance of experts [58, 64, 78]. Thus, the over-reliance on predicting a subject's performance on her experience with code artifacts may be a threat to locate expertise precisely. Second, current location approaches neglect the constraints for experts to coordinate such as neglecting time availability and geographical distance [62]. Finally, there is a lack of empirical evaluation in real-world context, and a few studies build the ground truth of expertise to evaluate their approaches by cross-validation [7, 88].

Based on our review, we provide and discuss following guidelines for future studies in expertise location. Particularly, we suggest to bridge the gap between early cognitive studies and location approaches, i.e., include supporting data of experts performance rather than only considering their activity record. Supporting data includes feedback from other peers and the organization or community since these sources have watched or collaborated with the expert when performing their tasks. Besides, while designing the expertise location systems, the designer needs to beware of the *paradox of expertise* [27], i.e., experts may be biased by their ability and their previous experience. Depending on the nature of the task, such as

training and expertise sharing, the candidate with highest expertise may not always be best choice.

The remainder of the paper is organized as follows. Based on several studies in cognitive psychology and neuroscience, Chapter 2 introduces the background on the nature of expertise. The protocol for conducting the survey and the evaluation matrices that we applied for all the studies in our literature repository are presented in Chapter 3. In Chapter 4, we present the result of our survey includes the major findings we mentioned above and comparison between the same type of expertise research. We provide a discussion of future expertise research direction and limitation of this work in Chapter 5. Chapter 6 concludes our study.

Chapter 2

Background

Lew Platt once commented in the 1980s:"If only HP (Hewlett-Packard[1]) knew what HP knows, we would be three times more productive," His words indicate that the problem of fully utilizing all expertise within an organization is desirable but hard to achieve for large organizations. For a software development organization, failing to identify and select proper expertise is a threat which lowers productivity, and further overburden central individuals (*expertise escalation[56]*), e.g., core product managers. As software industry growing, software engineering is vital for human beings' daily activities, and the failure of software may further threat the quality of our lives, which may lead to hazard at society level [87]. Locating and then utilizing an organization's own *intangible human resources* is a challenging two-steps (location and utilization) process for all software organizations and communities [13]. Extracting specific domain expertise from individual talent among these organizations is the critical first step.

[1]https://www.hp.com

2.1 Nature of Expertise and Expert

To locate expertise precisely, first, we need to have a clear understanding of the nature of expertise and the characteristics of who have them. According to the view of a psychologist, Ericsson et al. define expertise as the "*characteristics, skills and knowledge*" based on comparison between experts to novices and less experienced people [30]. Their definition indicates that people with related expertise would display better performance than those without it on a regular basis.

> "Expert: one who is very skillful and well-informed in some special field." -Webster's New World Dictionary
>
> "Experts are people who produce clearly above average (outstanding) performance on a regular basis." -Cambridge handbook of Expertise and Expert Performance

The above two definitions of expert are from Webster's New World Dictionary [40], and Cambridge handbook of Expertise and Expert Performance [30] respectively. Both of them suggest a relative approach to define expert by comparing experts performance with novices. However, as previous research suggests, there is another approach to identify experts by studying genuinely exceptional people, with the goal of understanding how they perform [20].

In addition to Ericsson et al.'s view, from a neuroscience perspective in Bilalić's research on explaining the nature of expertise [11], Bilalić classifies the expertise into categories. There are three major types of expertise when human beings are referring to the skills and specialties that expert masters:

- *Perceptual Expertise*: This type of expertise that predominantly rely on the information

directly come from human biological senses. For instance, the radiologist example mentioned in Bilalić, who could enable her visual expertise while detecting the problem of the patient through an X-ray photo.

- *Cognitive Expertise*: The second type of expertise focuses on the memory engagement and mental simulation, rather than information gathering process in the perceptual expertise, such as mathematical calculation skills. Particularly, the state-of-the-art research on software engineering expertise (for example, [7, 22, 35, 57, 60, 93]) focuses on this type of expertise when looking for programming experts.
- *Motor Expertise*: the last type of expertise is the muscle and body control ability, such as dancing, sports, playing music and other general movements. An example expert of this type is a professional basketball player who can hit shots from long range in a higher chance than an ordinary novice.

In addition, if we look at a definition of expertise by software engineering researcher by Mockus and Herbsleb [60]:

> Expertise is defined as the skill of an expert, and, if interpreted quantitatively, reflects the degree of the ability of a person to perform certain task.

The concept of expertise can be interpreted as a quantitative value, but in reality it is very complicated to measure, for example, early studies in cognitive science employed eyeball movement tracking to measure the information processing ability [37].

Another general expectation of experts is that they utilize their expertise and previous experience to handle any unexpected situation and outcome [30]. Since experts have already acquired enough knowledge as a structured system in their *long-term memory*, they would quickly adopt the situation in their field, and they also would be able to automatically provide and recognize reasonable solutions rather than figure them out.

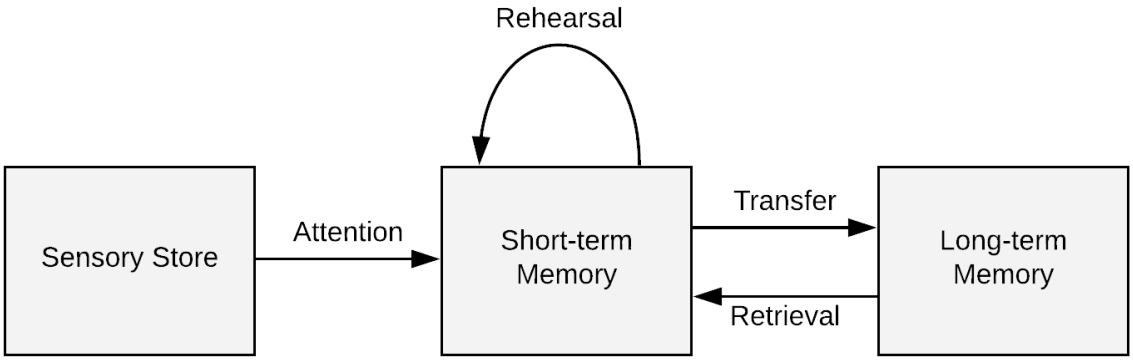


Figure 2.1: The different memory sections, and the process of transferring and retrieving relation between sensory store, short-term memory and long-term memory

In this study, we would cover basic approaches to locate expertise (absolute and relative), comparing their measurement of expertise, and finally evaluate these expertise location studies. Moreover, we attempt to summarize literature that considers the above two factors of experts, i.e., information processing ability, and previous experience.

2.2 Memory Engagement and Expertise

As this study particularly focuses the expertise and expert location in the field of programming and software engineering, the most critical type of expertise is the *cognitive expertise*, though the other two types could also be involved in the software production. The connection between memory and expertise is very tight due to our memory storage system. Many studies have been conducted on the structural model of human memory, and memory/expertise retrieval mechanism. The most related ones are the Chess Experiment for Chunking Theory [17, 25, 37] and the *Amnesia Patient Experiment* [21].

Human beings receive raw information and store it at the sensory store, but only if the information got human attention, the information would be processed to the *short-term*

memory, which is not everlasting and typical only stay for about 18 seconds without rehearsal [68]. Further, through several iterations of rehearsal, information in short-term memory would be transferred to *Long-term Memory* which is the *Synaptic Consolidation* process [28], and the information is hard to forget through a forgetting process. Therefore, the Long-term memory reflection is an essential indicator of having the expertise, and cognitive literature has also supported it:

> Hence we can say that one part of the grandmaster's chess skill resides in the 50,000 chunks stored in memory, and in the index (in the form of a structure of feature tests) that allows him to recognize any one of these chunks on the chess board and to access the information in long-term memory that is associated with it. The information associated with familiar patterns may include knowledge about what to do when the pattern is encountered. Thus the experienced chess player who recognizes the feature called an open file thinks immediately of the possibility of moving a rook to that file. The move may or may not be the best one, but it is one that should be considered whenever an open file is present. The expert recognizes not only the situation in which he finds himself, but also what action might be appropriate for dealing with it.

As mentioned by Simon in his masterpiece, the Science of the Artificial (pp. 89), when referring to experts, we expected that they could retrieve their similar experience or expertise to solve the problem at hand [77], and De Groot lead studies on the expertise retrieval from long-term memory and purposed the chunking theory [17, 25, 37]. Early in 1965, De Groot conducted the chess experiment to explore the performance difference between chess experts and novices. Each participant had 5 seconds of viewing and remembering a chess board with positioned pieces and then trying to re-position all pieces on the board later. However, in short, experienced expert chess players could recall more pieces and make fewer mistakes. As the progress made in neurobiology, researchers realize that information is stored in memory

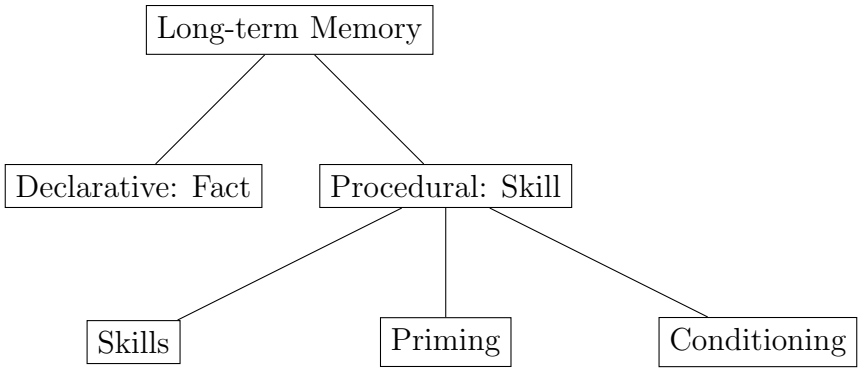


Figure 2.2: Knowing that (declarative) vs. Knowing how (procedural): the long-term memory model from Cohen and Squire [21]

as chunks [17]. Chase and Simon in their following study of chess experiment, they explain the superior performance of experts with the better functioning ability of chunks. Experts could retrieve more and larger chunks from their memory of specialized field. In addition, a later study [69] suggests that once experts encounter a new/unexpected situation in their specific domain, and then experts would automatically active their domain-specific expertise in their long-term memory, which evinces the second expectation for expert in section 2.1.

Further, it is worth noticing that, according to empirical results, there is a difference between "knowing what" and "knowing how" in human long-term memory. In 1980, Cohen and Squire conduct the study on Amnesia patients to identify which part of patient's memory or confirm whether totally had been impaired [21]. However, the result suggested a detached structural model of long-term memory. The performance of patients suggests that Amnesia patients' "knowing what" ability is impaired, but their ability to acquire or recall ("knowing how") skills still remain. For example, they could learn or recall how to ride a bicycle. Their study creates an initial structural model for long-term memory, and our study would refer this series of model to analyze long-term memory reflection from surveyed literature.

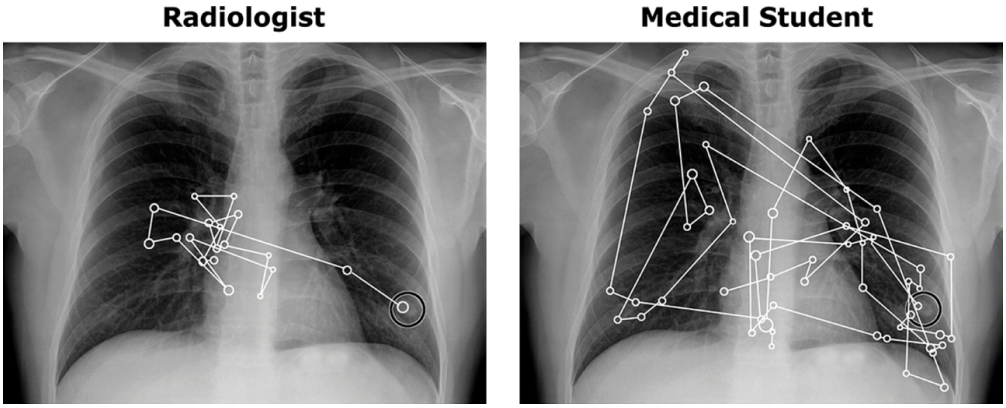


Figure 2.3: The Radiologist Example [11]. It Shows the Difference Between Experts and Novices When Processing Visual Information

2.3 Expertise Location in Software Engineering

Chunking theory can also be supported by the expert developer's performance in software engineering activities based on empirical results. McKeithen et al. have conducted a study on the different performance between programming novices and experts [58], and it empirically confirmed that experts have superior performance throughout experiments, and also they confirmed that chunking theory still holds for programming performance, as experts can recall more programming semantic concepts while solving the task. In addition to information processing ability, McKeithen et al. also mentioned that the organization of computer programming knowledge in experts mind are remarkably similar, but not identical. However, following up studies on expert performance fail to apply this conclusion, which utilize expert similarity to locate experts.

A software system is an artifact that requires tremendous effort to develop due to its complex nature, and there is no one for all solution to increase software productivity [14]. Due to the complexity of software engineering, there are several sub-domains of expertise. Moveover, FOSS are increasingly used as components in other software artifacts. Hence, identification of the specific domains experts is particularly hard in the context of freely formed OSS teams.

It is a long discussed question to define and measure expertise in a quantitative approach systematically. A quantitative approach to measure the expertise level of chess players is tracking their eyeball movement, since expert has the ability to automatically exclude irrelevant information, i.e., chess master has the capacity to focus on the most critical part of board for wining the game, but novice is not able to focus on the relevant part for planning wining strategy. Similarly, expert radiologist only needs a few glances to understand an X-ray with very few eye focus movement, but novices such as medical students require move time and more effort for a same process [11]. There are several pioneer studies for locating experts in software developing process [56, 57, 60, 67]. As McDonald and Ackerman found in their studies [56, 57], one of the most effective way in engineering practice identifies expert according to developer's previous experience, which could be reflected by two indicators: a developer's historical artifacts, and people who she have worked with (her professional social network). Their work has been followed up by other researchers in software engineering, and most of their work focus on extracting the higher precise and accurate expertise identification over historical artifacts such as code [7, 60, 73].

Nevertheless, these expertise location systems are considered biased due to only enabling public information, and user reputation could be easily manipulated by faking data, or operating multiple accounts. However, data provided through these user-generated content enables more data sources than previous methods, and as the open source software is increasingly involved in other software engineering process, either in academia or commercial industry. The transparency provided by OSS collaboration sites could afford more activity information while locating expertise [23], and distributed collaboration approach also benefit from an automated expertise identification when the team is not familiar with each other. On the other hand, precisely locating the best knowledge providers on knowledge sharing can also smooth the process with a higher performance.

However, in addition to mining historical artifacts, there is emerging emphasis on social

networking functions for online software collaboration sites [23] and Q&A knowledge sharing sites [84]. To encourage participation and reward contributors, these platforms provide endorsements, ratings, stars, reputation point and other rewarding systems to acknowledge the achievement of a software developer. This type of acknowledgement is certified by peers of the contributor who use the same platform such as the questions acceptance in Stackoverflow, or the system automatically generate the certification based on the records such as Coderwall badges.

There are a few studies [70] that enable these various data generated by other users. The qualitative results have suggested it is promising in hiring software developer occupations even though interviews are necessary. We will present our detailed systematic survey results in the chapter 4.

Chapter 3

Approach

After reviewing early cognitive studies on expert performance, we present this study of systematic literature review which explores the current practice of expertise location in Software Engineering. First, we illustrate the approach and protocol to conduct the review. This review is based on guidelines provided by Kitchenham and Brereton [46]. In following sections, we specify the goals for each research question, and then provide the six-step-process in details for our review, the keywords and domains for literature searching queries: the list of literature searching engines, the standard for conference filtering, and finally the evaluation matrices for literature.

3.1 Research Questions

There are different types of experts in the field of software engineering practices. When organizers or managers intend to maximize the benefit of specific domain experts for the team goals, they need to understand the expertise of each member in the group. To understand the capability of an organization, one of the most intuitive and efficient way is locating experts

from their previous experience [57]. Due to the popularity of online open source communities, we expect to employ more data sources to generate suggestion for locating different types of experts through rich public activity data. In this study, we survey the nature of expertise in the field of software engineering, and another goal is to review approaches from state-of-the-art expertise location research. Therefore, we purpose the following two main research questions:

RQ1: In the field of Software Engineering, what are the measurable characteristics that distinguish experts from novices or less-experienced subjects?

By answering the first research question, we target to conclude the characteristics that differentiate experts from less-experienced and novice developers. In addition to studies included in the background section, we also expect to learn through surveyed studies to find if there were other criteria to identify expert developer.

RQ2: How do state-of-the-art expertise/expert location approaches and systems compile these characteristics and then locate expertise?

From the second the research question, we intend to investigate data sources and expertise models that each study adopted to determine their experts under various sub-domains. However, there are various and complex sub-domains under software engineering. When searching for various expertise location methods, we intend to examine if the model can be applied to OSS community, or if OSS public data could contribute to improve the locating quality (e.g., precision and recall) while including additional public data.

Through answering these two research questions, we expect to summarize the approaches for locating different types of expertise/experts in software engineering, and provide directions to design future expertise location systems which leverage large public datasets from OSS and knowledge sharing sites. Further, use these experts as role models to guide novice software

developer for their careers [81].

3.2 Literature Survey Framework

Our literature search approach is inspired by following literature survey studies in software [31, 81]. Before starting the research process, we set up the targeted publication venues. In order to ensure the focus and quality of the literature that we are studying for constructing the initial sample, we filter out the conferences and journals other than A-ranks according to ERA[1], Qualis[2] and CCF[3], these three rankings systems. In addition to the initial searching process, we performed one round of citation searching (snowball sampling) after we constructed the initial sample. Further, at the end of sample construction, we manually added several pieces of relevant literature to our literature repository.

The overall literature selection process is as follows:

1. Searching digital libraries: we query digital libraries, and then store the query result at a local paper repository which is managed by a free and open sour-source reference management tool, Zotero[4], and it manages bibliographic data of the paper and also related copies of the paper, such as PDF files. Moreover, we employ its chrome extension, Zotero Connector[5] to transfer the search result from web to the local paper repository.

2. Title, abstract and keywords filtering: since our query contains several generic terms in the domain of software engineering, the searching result is not promising in the recall. We perform a first level of filtering on title, abstract and keywords to check if

[1]http://libguides.newcastle.edu.au/researchimpact/eralist
[2]http://www.capes.gov.br/
[3]http://history.ccf.org.cn/sites/paiming/2015ccfmulu.pdf
[4]https://www.zotero.org
[5]https://chrome.google.com/webstore/detail/zotero-connector/ekhagklcjbdpajgpjgmbionohlpdbjgc?hl=en

the studies are relevant to our major objectives based on meta-data. From this step, we exclude most irrelevant papers to our research questions, which reduces the cost of the study.

3. Introduction and conclusion analysis: then we analyze the starting and ending sections of the literature to identify their primary research goals and contribution. This part of analysis could enable us to one step further to validate whether the purpose of the subject is relevant to our research questions.

4. Citation search: for studies found relevant after all previous steps, we search on Google Scholar which aggregates literature across different resources. Then we perform the same analysis on meta-data, introduction, and conclusion to decide whether to include these studies. At this step, we dismiss the filter of Rank A conferences but include literature from all other publication venues as long as the paper is relevant to the objective.

5. Manual addition: we also include a few studies that have solid contribution to this field, especially answering our research question but not covered by the query result at the end of the study.

6. Full literature reading: at last, if a paper is included after all previous steps, we perform a comprehensive reading throughout the paper, but also decide whether to exclude the literature in the last step.

According to our research questions, we restrict our studying fields with the range of Software Engineering, Human-Computer Interaction, and Computer Supported Collaborative Work. By limiting the conferences that we are studying, we not only limited our research focuses based on RQs, but also exclude papers that are not in English or not published in a peer-reviewed venue. As a result, based three publication venue ranking systems that we referred,

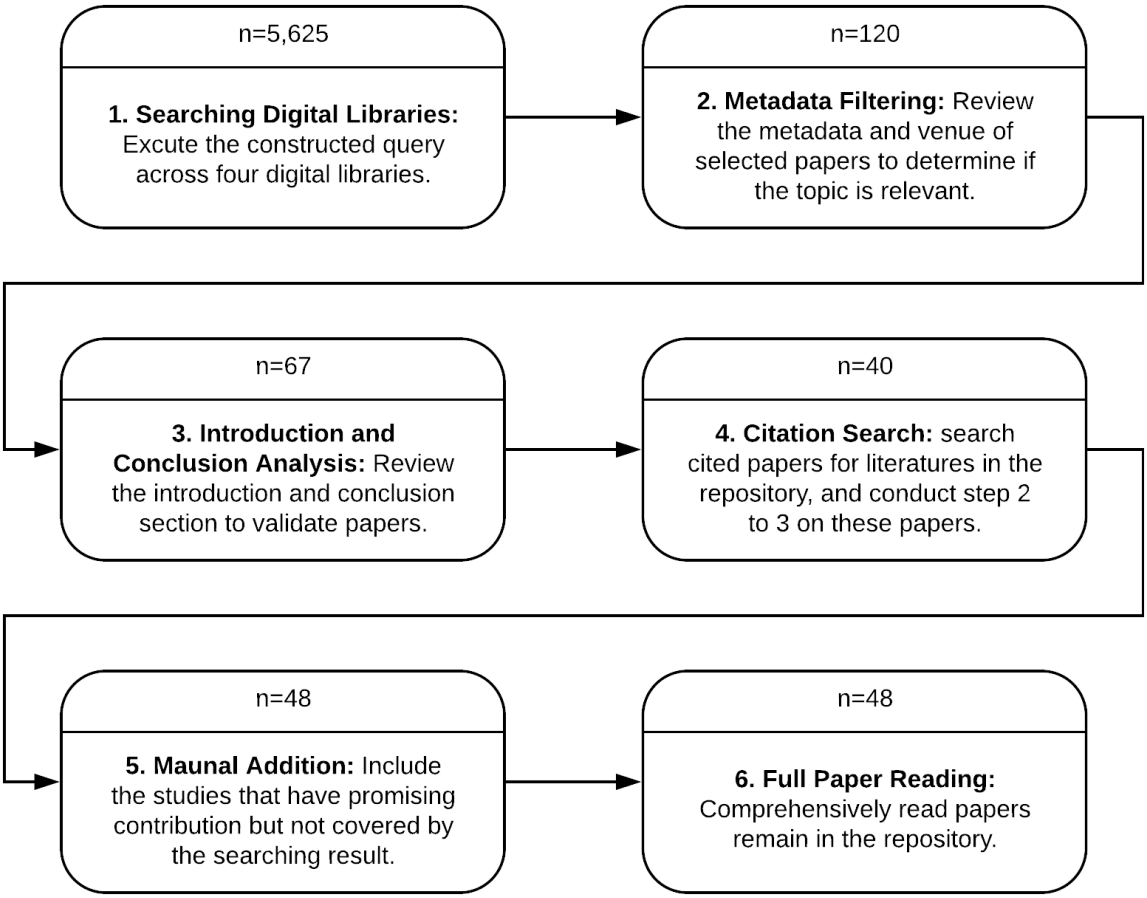


Figure 3.1: Literature Selection Process

Item	Keyword
Domain I	software, system, program
Domain II	development, develop, engineer
Expertise	expertise, expert, skill, knowledge

Table 3.1: Keywords for Constructing the Searching Query

```
{
  ("software" OR "system" OR "program")
  AND
  ("development" OR "develop" OR "engineer")
  AND
  ("expertise" OR "expert" OR "skill" OR "knowledge" OR "ability" OR
  "talent")
}
```

Figure 3.2: Query Performed for Searching Literature

the rank A conferences under these categories are: ICSE, FSE, ASE, CSCW and CHI; the rank A journals are: IJHCS, TOCHI, TOSEM, TSE, and CSCW.

We determine keywords that we would include to perform the literature search (see Table 3.1) upon RQ1 and RQ2. The keyword list has two major fields that we are interested in this study: domain of software engineering, and expertise location. The first set of keywords (Domain I) indicates software artifact is the subject that we concern and its synonyms. The second the set of keywords (Domain II) indicates that we are interested in expertise involved in the production procedure of software. The last set of the keywords contain the main subject of this study, expertise, and its synonyms.

The query based textual search expression is an effective mechanism to retrieve literature through digital libraries. Based on the keywords in the Table 3.1, we constructed a searching query connected by binary operators, which is supported by most literature search engines (see Figure 3.2).

After determining the query, we select digital libraries to conduct the search. Our selection criteria are based on following: It should include major publication venue in Software

Conference	Searching Results
CSCW	278
ICSE	1451
CHI	2224
ASE	84
FSE	70

Table 3.2: Searching Result of Conference Publications

Journal	Searching Results
CSCW	505
IJHCS	811
TOCHI	69
TOSEM	92
TSE	41

Table 3.3: Searching Result of Journal Publications

Engineering, HCI, and CSCW in English, support binary operators in queries, and provide full paper access. Therefore, we selected four digital libraries to conduct our search: ACM Digital Library [6], IEEE Digital Library [7], Scopus [8], and Springer [9].

3.3 Literature Sampling

Searching Digital Libraries: We used the query presented in Section 3.1 to retrieve an initial sample of papers, which was executed on Mar 19, 2018 across four digital libraries, and results are ordered by relevance according to each digital library. We also used the Rank A conferences in Section 3.1 as a filter to limit result amount. All queries are performed on meta-data indexing of the literature for consistency (The meta-data of a paper includes its title, abstract, and keywords). See Table 3.2 and 3.3 for initial sample by publication venue from aggregated searching result. Noticeably, since we employs a reference management

[6]https://dl.acm.org/
[7]https://ieeexplore.ieee.org/
[8]https://www.scopus.com/
[9]https://link.springer.com/

Conference	Initial Sample
CSCW	12
ICSE	39
CHI	15
ASE	10
FSE	6

Table 3.4: Amount of Papers for Each Conference Venue in the Initial Sample

Journal	Initial Sample
CSCW	10
IJHCS	15
TOCHI	1
TOSEM	4
TSE	8

Table 3.5: Amount of Papers for Each Journal Venue in the Initial Sample

software to manage our paper repository and bibliographic data, it automatically to filter the duplicates of papers as we transfer the searching results into the repository in the software. Thus, there is no such a phase in our approach that removes the duplicates.

Meta-data Filtering: Then based on the searching result, a researcher manually inspects the meta-data of each result to determine whether the literature is relevant to the topic. At this phase, we exclude papers that not focus on software, e.g., expertise sharing over a medical team, papers on expert systems, papers of keynote speeches and presentation notes, and also duplication. As a result, we generate an initial sample of 120 candidate papers from 5 conference venues and also 5 journal venues. See Table 3.4 and 3.5 for numerical details by venues.

Introduction and Conclusion Analysis: In this step, we focus on the research method, the contributions, and the application scenarios of studies include. We exclude 53 papers which are not fitting our objectives in the research questions, for reasons such as an observational study without purposing a locating approach or system.

Citation Search: After the previous step, we conduct one level of citation search (snowball

sampling) on papers in the repository. We mainly focus on studies conducted by authors already in the repositories and researches with high impact (high volume of citation count provided by Google Scholar). We initially relied on the citation counts from the original digital library where paper was found, but the result is not promising as the original library may be in lack of counting citations from publication venues are not included in this library. Therefore, we apply Google Scholar to search the paper again and employ the citation count to determine our initial and very brief assessment on research impact. From this step, we include another 6 studies into our paper repository, and also we do not filter papers from publication venue standard that we used at the second step.

Manual Addition: There are a few important studies that are not covered by our searching query, neither could not be reflected through snowball sampling. These studies either are not covered by our query since it is published in a small conference, or are indirectly related to our research focus in this study. Therefore at the end of the sample construction process, we manually add a few studies based on expert recommendation that have promising research objectives, or provide future directions in the field of recommending expertise. These studies are most aggregating data across multiple collaboration sites. We will provide a detailed analysis in the result section.

Full Paper Reading: In the last step of the sampling process, we start our analysis process while still be open to exclude studies that not focus on our research questions. As a result, we did not exclude any papers at this stage. In the following section, we will provide the evaluation matrices that we adopted during the full paper reading process.

Item	Purpose	Definition
Title	Overview	What is the title of the literature?
Year	Overview	When did the literature publish?
Publication Venue	Overview	Where did the literature publish?
Publication Type	Overview	What is the type of the literature (journal, conference, demo, or book)?
Authors	Overview	Who are the authors of the literature?
Research Type	Overview	What is the research approach of the literature (empirical, case study or conceptual)?
Expertise Domain	RQ1, RQ2	Which domain can the expertise being apply to? Software coding/testing/design etc?
Expertise Model	RQ1, RQ2	What is the mathematics or theoretical model of the expertise?
Theory base	RQ1	Are there any theoretical background for this model (e.g., from cognitive science/engineering/novel)?
Memory Engagement	RQ1	Does the expertise model reflect any memory engagement associating with expertise?
Application of Studies	RQ2	How does the expertise/expert location system improve the field of software engineering?
Granularity	RQ1, RQ2	What is the granularity of the expertise model (element/method/file/project)?
Future Work	Direction	What is future work of this study, particularly on expertise location?

Table 3.6: Evaluation Matrices for Reviewing Papers

3.4 Evaluation Matrices

While conducting the full paper reading, we set up the evaluation matrices to collect items that we are concerned according to our research question. We listed the following 13 items extracted from each literature in our paper repository (See Table 3.6).

Research overview: There are 6 items in this category. We included the meta-data of the publication in this category such as title, year, publication venue, publication type, authors and research type, especially authors of research which used for extracting primary studies in later sections. In addition to meta-data, we also identify the research types (*e.g.,* conceptual, empirical, case study, ethnographic *etc.*) for each paper that we reviewed. Therefore, we could summarize the suitable research methods for specific research questions in expertise studies.

Expertise Domain: As we mentioned in the background section 2.3, software engineering is a complex subject with multiple sub-domains. Through collecting expertise domain, we intend to investigate the status of expertise location techniques in a specific area, for example, bug report assignment, merging conflict resolution, and software fault location.

Expertise Model: We use this item to capture the conceptual or mathematical model for locating expertise. For example, Servant and Jones [74] use a mathematical model of

speciousness of a fault and *recency* of changes on a specific line of code to determine the expertise of a developer to fix a bug. Further, an expertise model can also be represented by a very high-level conceptual model such as [92] uses a four-layer model to classify knowledge and expertise of functionality in an application. We intend to summarize the model that locates, or even quantifies expertise for software engineers in a specific domain respectively.

Theory Base: We also intend to collect any theory base behind the expertise model that each study adopted. The theory can be purposed within the study based on author's suggestion, or a field study conducted before, or referred from engineering or management literature. Theory bases evince the expertise model that adopted by these studies, and are applicable for a larger scale.

Memory Engagement: Based on many studies that were referred to in the background chapter, the cognitive expertise for software engineering task is rooted in a developer's long-term memory. From this item, we intend to investigate the long-term memory reflection of expertise models, *i.e.,* whether an expert determined by an expertise model was capable of reflecting information processing ability such as being able to recall more related memory for the subject.

Application: This item focuses on the application and contribution of this study. We collect this item mostly according to authors suggestion and proposal in their paper, which are usually represented in the introduction, discussion, and conclusion sections. By summarizing this item, we intend to explore our views on implications for expertise location techniques in software engineering, such as in hiring, training, and so on.

Granularity: The granularity of the study mainly focuses on the scale of expertise could be reflected by the model. If an expertise model utilizes the methods level coding behavior to assign developer for implementing a program method, then this model would be not applicable to determine this developer's overall knowledge on the project which contains the

method.

Future work: Suggestions for future work or directions can usually be found in discussion and conclusion sections. Although each study has its own very different purpose, from this survey, we aim to analyze and summarize the suggestions from each sub-domains, and overlook a high-level future direction of expertise location approaches.

We analyze papers in our literature repository based on the evaluation matrices listed above but not limited to. There are several cases that some of the item are not applicable to a particular study, but the overall the evaluation matrices is actionable for the majority of the studies in the paper repository.

Chapter 4

Results

In this chapter, we present the results of the literature survey based on the literature repository that we constructed in Chapter 3. The literature repository is stored in a local computer of the author and managed by a reference management software, Zotero. We extract and organize the results based on our evaluation , and we summarize five main different categories of expertise studies. In the following subsections, we first present the overview and comparison of the literature in our paper repository, and then we summarize and highlight the most impact research in each category. Finally, we summarize findings from our review.

4.1 Overview

Year Distribution: Earlier research on expertise focus the cognitive science of how experts think and act. Though several directional studies [56, 57, 60] emerged around 2000, most follow-up studies of automated location systems were published in the following decade. To the best our knowledge, after the first study [7] applied machine learning techniques in locating expertise for bug report assignment, there are more studies that apply the statistical

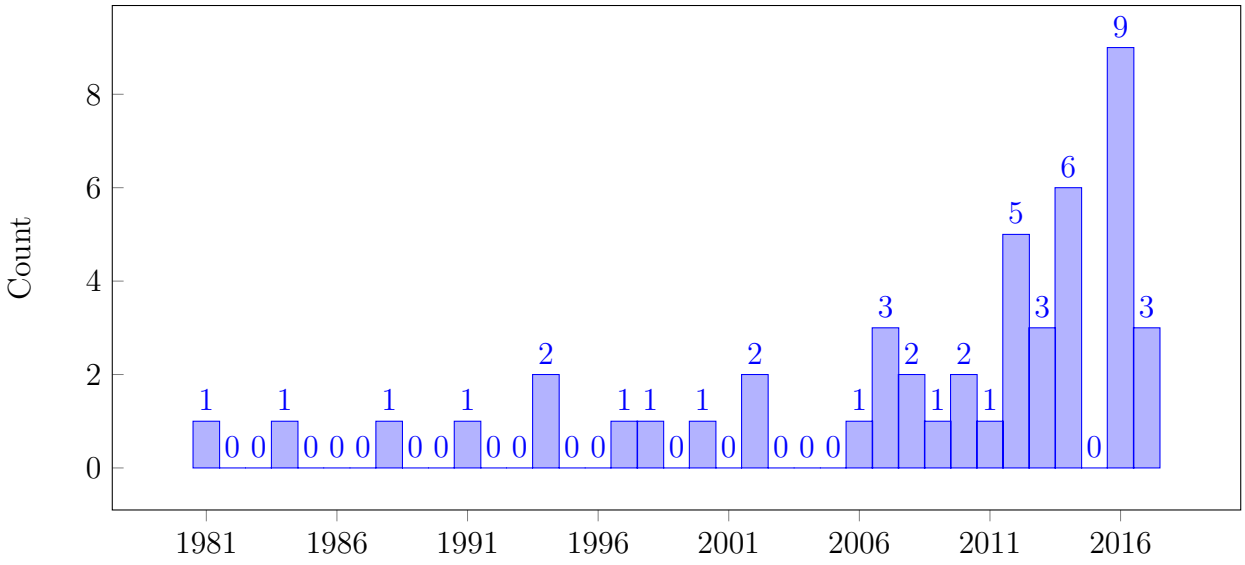


Figure 4.1: Publication Year Distribution of Papers in Literature Repository

models to analyze human factors in software engineering [88, 91]. In our paper repository, most studies were published in 2010s (See Figure 4.1).

There are 20 journal publications and 28 conference publications respectively in venues of CSCW, Software Engineering and a few in HCI. Software Engineering has the most published papers in this repository (28 out of 48).

Impact/Citation Count: The average citation count of these studies is 99.4, and there are 10 studies have over one hundred citations. Hence, we argue that studies in our literature repository represent the research on expertise locations and automating this process.

Research Type: There are 23 empirical studies, 18 case studies, and 7 conceptual studies. Noticeably, one study conducted observation in the field to build a ground theory for expertise location approaches in software engineering practice [56]. Among these studies, 23 of them only use quantitative approaches to analyze their collected data, and 11 studies only apply qualitative approaches in their studies. Further, 7 studies apply the mixed method to evidence their conclusions. According to our literature repository, we found there is a variety of research methods based on their research focuses. Studies that mine historical

Year	Reference	Year	Reference
1981	McKeithen et al. [58]	2012	Dabbish et al. [23]
1984	Soloway and Ehrlich [78]	2012	Hanrahan et al. [42]
1988	Pinto and Soloway [64]	2012	Servant and Jones [74]
1991	Koenemann and Robertson [48]	2013	Maalej and Robillard [53]
1994	Davies [24]	2013	Ackerman et al. [3]
1994	Stanislaw et al. [80]	2013	Yarosh et al. [90]
1997	Waterson et al. [86]	2014	Fritz et al. [36]
1998	McDonald and Ackerman [56]	2014	Vasilescu et al. [84]
2000	McDonald and Ackerman [57]	2014	Maalej et al. [54]
2002	Mockus and Herbsleb [60]	2014	Ye et al. [91]
2002	Nardi et al. [61]	2014	Ley et al. [49]
2003	Chevalier and Ivory [19]	2014	Bergersen et al. [10]
2006	Anvik et al. [7]	2016	Chan et al. [16]
2007	Minto and Murphy [59]	2016	Xu et al. [88]
2007	Anvik and Murphy [6]	2016	Thongtanunam et al. [82]
2007	Reichling et al. [67]	2016	Yu et al. [93]
2008	Yang and Chen [89]	2016	Costa et al. [22]
2008	Schuler and Zimmermann [72]	2016	Rahman et al. [66]
2009	Lin et al. [50]	2016	Hannebauer et al. [41]
2010	Kläs et al. [47]	2016	Sarma et al. [70]
2010	Fritz et al. [35]	2016	Greene and Fischer [39]
2011	Gottipati et al. [38]	2017	Lin et al. [51]
2012	Pipek et al. [65]	2017	Saxena and Pedanekar [71]
2012	Bednarik [8]	2017	Wang et al. [85]

Table 4.1: Primary Studies in the Literature Repository

artifacts usually apply quantitative method to model expertise [7, 59, 60, 84], and studies explore the expert characteristics and commonalities in sharing expertise activities usually adopt qualitative research method [18, 48, 49, 86].

4.2 Primary Studies

In the final literature repository, we read the paper in full and then extract necessary data from each paper. Therefore, in the end, we identified these studies as primary (see Table 4.1) based on their focused expertise domains and correspondence to our research questions.

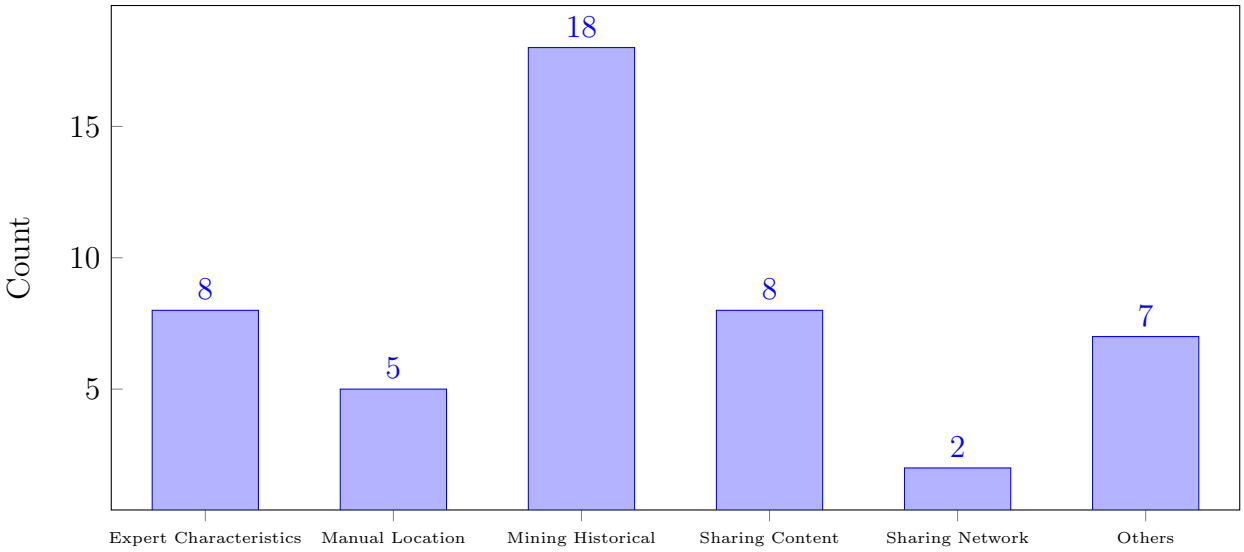


Figure 4.2: Publication Category Distribution of Papers in Literature Repository

In our literature repository, we identify two board categories of expertise research: 1) identify the characteristics of experts, 2) locating experts.

There are several early studies and few recent studies intend to empirically identify the characteristics of experts in software engineering by monitoring their performance, see Table 4.2. There are studies explore the information processing ability of experts [58], the strategies that experts adopted in programming comprehension [48], and validation expertise via monitoring experts performance [10, 24].

There are several research investigated the practice of expertise location in software engineering. These studies focus on the common practice that manually locates experts (see Table 4.3). There is a study identifies the expertise sharing network structure in software development organizations [86]. Moreover, there is a study identifies a set of practical strategies for locating expertise, which provides heuristics for automated expertise location systems [56]. Finally, there is a recent study that revisited the practices of expertise location in software organizations but found asking colleague is the most common practice, and automated location tools are hardly used [90].

Year	Reference	Approach	Findings
1981	McKeithen et al.[58]	Empirical Study	Experts in programming perform better in retrieving and recalling related information; experts' knowledge representations are similar.
1984	Soloway and Ehrlich[78]	Empirical Study	Empirically identified two types of knowledge in programming: plans and rules, which are heavily related to mind simulation and information processing.
1991	Koenemann and Robertson[48]	Empirical Study	When understanding the program, experts search for relevant information in the code, and only employ a bottom-up strategy for entire program comprehension when their hypothesis failed.
1994	Davies[24]	Empirical Study	As the expertise growth for programming developers from lower to higher, when given a programming task, their reaction time and the number of errors made tend to be lower comparing to novices.
1994	Stanislaw et al.[80]	Empirical Study	Employ time factor, time of complete the programming task, and multitasking ability, the variety of programming skills such as the number of programming language known, to quantitatively measure expertise.
2014	Maalej et al.[54]	Case Study	Programmers comprehension strategies depends on the context, and they would avoid to understand whenever possible, similar to work of Koenemann and Robertson.
2014	Bergersen et al.[10]	Empirical Study	Bergersen et al. build testing model to systematically measure and valid the expertise level of programmers in Java. It validates expertise the quality of answers and responsive time in series of programming tests, and also self-reported data of the experience and programming motivation.

Table 4.2: Primary Studies for Expert Characteristics

Year	Reference	Approach	Findings	Future Work
1997	Waterson et al.[86]	Case study which visited an IT company several times and conduct interview based qualitative research with key personnel in the organization	They identified the manual knowledge sharing network structure in a software development team	Technology can be applied to knowledge sharing process in the software development. Purpose a need of conducting case studies upon different size of organizations to validate their structure.
1998	McDonald and Ackerman[56]	Conducted a field study in midsize IT company.	They found three major heuristics to locate expertise in software development practice: everyday experts, through an expertise concierge and mining historical artifacts.	Automated systems are needed to support situations in expertise location process such as the escalation behavior.
2012	Pipek et al.[65]	Longitudinal case studies in three organizations and summarize the frameworks for designing technologies for supporting expertise and knowledge management.	In expertise sharing of organizations, practice leads to technologies innovation. While integrating new tech, the new design should not disrupt the current ecosystem.	Follow the heuristics and carefully design new technologies for supporting expertise location and sharing activities.
2013	Yarosh et al.[90]	They conducted a diary and then interview study on professional software developers	They explore the expertise location practice, and they found in most cases (76%) asking a colleague is the common practice to solve a problem, and tools are not helpful.	Time issue in the expertise (help) location activity. They found 15% of the tasks involved in help-finding remained unsolved for longer than two weeks.
2016	Thongtanunam et al.[82]	Observe the code reviews of six software systems in the field	Majority of reviewers only contribute to one module on the system, and code reviewers with more reviewing experience are more significant in reducing the defects rather than with code authorship.	Future studies should majorly rely on using code review experience to determine best experts for reviewing the code.

Table 4.3: Primary Studies for Manual Location Approaches

Year	Reference	Domain	Approach	Evaluation	Future Work
2000	McDonald and Ackerman[57]	Problem Casebase	Analyzing task similarity to find best expert for the current problem	Not applicable	Empirical study with developers.
2002	Mockus and Herbsleb[60]	Activity unit (Experience Atoms)	Accumulate the experience on historical artifacts to determine expertise	Usability test	Use time for completion to evaluate the expertise required for task; "quantitative resume" for developer.
2006	Anvik et al.[7]	Bug report	Analyzing the Bug Report similarity and use machine learning techniques to recommend developers	Precision and recall	Empirical study with developers, additional source of information.
2007	Minto and Murphy[59]	Changes in historical artifacts	Apply file dependency and authorship matrix to produce expertise matrix, and finally support experts communication by locating expertise of task	Precision and recall	Empirical study with developers
2007	Anvik and Murphy[6]	Bug Report	Empirically evaluate the expertise location approaches for bug report assignment: mining code artifacts and "bug network".	Precision and recall	Looking for approaches other than precision and recall to evaluate the expertise location system
2007	Reichling et al.[67]	Support Expertise Sharing	Aggregation of feedback for previous work, similarity of previous tasks and personal information.	Not applicable	Suggestion for applying technology in expertise sharing activities
2008	Schuler and Zimmermann[72]	Code Artifacts	Measure expertise based on code usage data (calling)	Not applicable	Improving precision, combining with conventional authorship model, combining with task and bug reports, explore the relation between expertise and quality of work
2012	Servant and Jones[74]	Fault Localization	Aggregate the test cases results execution results with code modification history, and then generate the fault-to-developer assignment.	Empirically compare the WhoseFault with historical ground truth, naive methods, and other automated techniques.	Expand the WhoseFault tool with additional subjects and test cases, and also apply different mapping techniques for fault-to-developer assignment.
2013	Maalej and Robillard[53]	API Documentation	List a taxonomy of knowledge type in API documentation through a four steps mix-method approach.	Case Study on two programming languages	Systematically identify the knowledge types in documentation rather than empirically. Address the gap between information seekers and providers.
2014	Ye et al.[91]	Bug Report	Employ machine learning techniques to aggregate data such as program classes name similarity and bug fixing history, and generate a list of files which are relevant to the bug.	Precision based evaluation metrics	Additional data source such as code authorship, and empirical evaluation with other projects.
2014	Fritz et al.[36]	Code Artifacts	Apply the Degree-of-Knowledge model to assess the expertise of a developer depending on her interaction with IDEs and code authorship (modification history)	Case Study with developers at three sites by applying the model to find experts.	Application in bug finding. Longitudinal studies with developers. Additional data source such as documentation.
2016	Rahman et al.[66]	Code Review	Determine code reviewer by code authorship and also experience with the collaboration model such as pull-request model on GitHub.	Precision and recall, comparison with existing systems	Handling concurrent recommendation request
2016	Greene and Fischer[39]	Developer Profile	Retrieve developer related tags from GitHub activity history	Preliminary case study in two company hiring process.	Additional data source, longitudinal study in software practice, and evaluate the tool accuracy with manual extraction.
2016	Costa et al.[22]	Merging Conflicts	Past experience of developers over different branches, and also the dependencies of modified files.	Empirical study uses historical data and compares with the actual person performed merge.	Integrate with more other algorithms, analyze in finer grain level, evaluate the project with other projects.
2017	Lin et al.[51]	Source Code Knowledge	Retrieve code-specific knowledge such API entities to support IR and learning.	Precision and recall of querying on 6 software projects	Improve document representation, additional data source and domains.
2017	Wang et al.[85]	Crowdsource Assignment	Predict the skill improvement of crowd-workers based on their learning curve.	Not applicable	Finer grain of skill improvement, empirical evaluation with developers, from crowdsource to open source.

Table 4.4: Primary Studies for Mining Historical Artifacts

The majority of studies that automated locates expertise are designed by following the heuristic of mining historical artifacts (see Table 4.4). The domain of these studies are : Software problem solution, bug reports, code review, and code artifacts implementation and comprehension, among others. New techniques have been employed as they become popular in software engineering, such as machine learning and natural language processing. The most adopted method to evaluate these expertise research is precision and recall, which is well-known method for information retrieval [32]. Most studies mentioned a lack of empirical validation of their approach and purpose to aggregate more data source to improve the expertise model. There is study adopts the learning curve theory to measure the skill/expertise increases [85], but it still lacks evaluation validate the approach. We are not able to identify other expertise models that measure expertise based on early theory or empirical results based on expert performance [48, 58, 78].

Knowledge sharing sites are transforming the way of expertise sharing activities. Instead of locating the expert, within these platforms, finding the relevant answer which is somewhere online. Providing incentives for knowledge providers is a more focused and needed research topic, e.g., Vasilescu et al. claimed the gamification features made the success of popular knowledge sharing site as StackExchange [84]. Further, the keyword-based search engines are not able to precisely locate the best matching question, and therefore, techniques such as semantic searching and neural networks are introduced to help the online knowledge sharing activities [38, 88].

A few studies are visualizing the social network of personnel in the organization. These studies support the awareness of the social network of each in it and hence support the communication among teammates. However, these studies usually do not provide evaluations of their tools, and particularly in the real-world context. Further, we are not able to find recent follow-up studies in this category.

There are other studies of expertise in our literature repository, such as exploring the im-

Year	Reference	Approach	Findings	Future Work
2008	Yang and Chen[89]	Case Study with a mixmethod analysis. Use Bloom Taxonomy Matrix to represent a person's expertise.	A p2p network system can support knowledge activity to effectively share knowledge, and need to locate relevant collaborators effectively.	Investigate the dynamics for different social network, and also support collaboration.
2011	Gottipati et al.[38]	Semantic search engine for finding answers, and empirical study on precision and recall, and finaly user study comparing with conventional tools.	Based on the evaluation metrics of precision and recall, and the nDCG@2 score, their approach significantly improve the answer finding experience.	Automated approach that cluster similar questions to help seekers to find relevant answers, and conduct study with other software forum.
2012	Hanrahan et al.[42]	Case Study on Stackoverflow data	They have not yet found a strong correlation between average expertise of involved users and the duration of the time, but it is an initial attempt to model difficult of question	Conduct studies on other expert community, and use AI systems to capture critical factors of problem difficulties.
2014	Vasilescu et al.[84]	Empirical study with r-Help mailing list and StackExchange data	Participants from both communities are more active than those participated one, and participants from StackExchange response faster because of the gamification.	Interview studies with participants from different communities.
2016	Xu et al.[88]	Apply neural networks to find similarity between different questions in software engineering	Based on precision and recall (both lager than 0.8), their approach effectively link similar knowledge units together.	Support more data type such as code snippet and image.

Table 4.5: Primary Studies for Knowing Sharing Sites

Year	Reference	Domain	Application	Visualization
2002	Nardi et al.[61]	Contact Information	Support communication, and awareness of social network	Map-based visualization which clusters the people under same organization network (see Figure 4.7).
2009	Lin et al.[50]	Professional Social Network	Support awareness of social network, and visualize the expertise domain of each person. Highlight key person in the social network.	Multiple view of visualization depending on its purpose. Node tree based visualization for social network, geographic visualization for distance and availability (see Figure 4.8).

Table 4.6: Primary Studies for Expertise Network

Year	Reference	Type	Approach	Findings
1988	Pinto and Soloway[64]	Documentation for Novice	Case Study with novices and their usage of program documentation	Novice without requisite knowledge of the program needs support of documentation, and also the presentation of the documentation matters.
2003	Chevalier and Ivory[19]	Implication of Expertise Study	Empirical study with web designer at different expertise level	Web design tool should support different design strategies and remind the guideline of design for different levels of expertise respectively.
2010	Kläs et al.[47]	Implication of Expertise Study	Case Study with industry	Expertise measurement can be applied to defects prediction in software product.
2016	Chan et al.[16]	Implication of Expertise Study	Empirical Study with crowd workers	Correctly facilitation experts into the crowd can help innovation and creativity in their work.
2016	Hannebauer et al.[41]	Empirical study on location techniques	Empirical Study for comparing different automated code review assignment techniques	Automated approaches based on review experience generate better performance.
2016	Sarma et al.[70]	Profile Aggregation	Activity Data Visualization and case study with hiring personnel	Profiles from online platforms (GitHub and Stackoverflow) are effective to understand a candidate's past, but interview is still needed.
2017	Saxena and Pedanekar[71]	Profile Aggregation	Activity Data Visualization	Treemap visualization based on the tags provided GitHub and Stackoverflow.

Table 4.7: Other Primary Studies

plications of identifying expertise level [16, 19, 47], empirical studies on the performance of location techniques [41], and expertise profile aggregation [70, 71]. These studies are not typical expertise location approaches but we believe they benefit this study while we build the our knowledge on expertise in software.

In the following sections, we summarize several most influential studies based on their citation counts and contributions to the area of expertise location. We also report these studies in a perspective of the historical view for the development of expertise location systems and approaches.

4.2.1 Expert Characteristics in Software Engineering

McKeithen et al. 1981

In earlier studies, researchers confirmed that experts are generally performing better than average in their mastered fields [17, 25, 37, 77]. From a cogitative perspective, they found the reason is that experts can retrieve more chunks from their related memory while working on their expertise domain, which gives experts a higher information processing ability in their expert domain.

In 1981, McKeithen et al. conducted two experiments in their study. First, they tested the performance between experts and novices in computer programming tasks by letting participants to view coherent or scrambled computer programs for a 3-min recall period for 5 trials, and then participants were asked to recall everything after each trial.

The result suggests that subjects with higher expertise in computer programming, they can recall more lines of the program in either standard or scrambled versions (See Figure 4.3). Their conclusion from this experiment suggested that in the field of computer programming, experts are also superior in information processing, including gathering or recalling

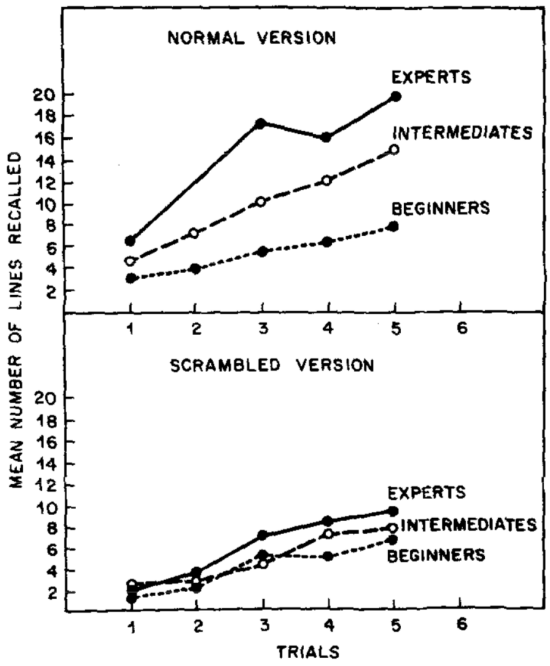


Figure 4.3: Performance Differences between Experts, Intermediates and Beginners in Experiment 1 (Recalling Lines of Programs) [58]

information which is similar to other fields.

In their second experiment, each subject was asked to organize the programming concepts in the form of a hierarchical representations for keywords in computer language such as, TRUE, FALSE, IF, WHILE and FOR, and so on. Then they aggregated the result into a multidimensional scaling configuration for computing the distance between each subject. Their result suggested that experts are more cohesive to each other as a group, whose knowledge construction, i.e., the semantic representation [12] are particularly alike, but intermediates and beginners are not cohesive as experts (See Figure 4.4).

This study has two major contribution: first, it confirmed the fact that in programming, experts are faster and precisely in processing information of computer programs which is a familiar result as in other field. Second, it found that experts are more similar while organizing their semantic knowledge structures about programming. Besides, expert subjects' knowledge structures have clustered a cohesive group.

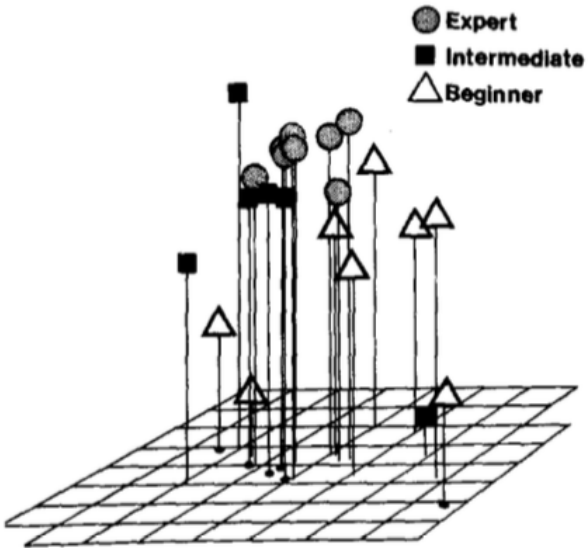


Figure 4.4: Coherence between Groups of Experts, Intermediates and Beginners. Measured by Multidimensional Scaling Configuration of Distance [58]

There are a few studies later on expert characteristics and performance, such as exploring expert problem-solving strategies [24, 48], quantifying programmer's skills [80].

4.2.2 Manual Expertise Location

McDonald and Ackerman 1998

In early days, practitioners locate expertise to solve problems that they could not solve alone, but over manual approaches without the assistance of automated software. In 1998, McDonald and Ackerman conducted a five-month field study in a medium-sized software firm to observe practitioner's behavior in manually locating expertise within their organizational settings [56].

According to their observation, three expertise *identification* approaches were summarized in the industrial practice. First, as they purposed the terminology in the paper, senior practitioners have difficulties in articulating how they know who knows a certain area, for

example, a quote from their study says:

> "You learn who's [the] most experienced in what areas. ...You just know." - Sherry

This expertise identification approach is the EVERYDAY EXPERTISE. However, this approach is not applicable to newcomers and sometime even to the senior member when they were unable to track everyone. The second type of expertise identification is HISTORICAL ARTIFACTS. The general philosophy of this approach is finding the person who has the latest authorship (change the artifact) of the artifact that related to solve the problem. However, this approach of identifying expertise is limited by the purpose or size of the change, which may falsely identify the expert.

The last approach is asking help from an EXPERTISE CONCIERGE. In an organization, the expertise concierge is the key personnel who has very elaborated social networks and mediates many requests for information including locating expertise. Other management studies use the term of information gatekeeper [4], information mediator [29], and information broker [63] to refer to the same role in an organization. An expertise concierge usually lead people who were looking for information and expertise to those who may have them.

After the identification process and if there were multiple choices, the practitioners would start the process of *expertise selection*. During this process, they tend to choose a person from their local social network and avoid routing to another department. Finally, the expert whom they chose to look for help might not always be the person who could offer the help due to series reasons, and this is when *escalation* happens.

This study is critically important to other studies of expertise location. It purposed three main strategies to identify experts, and two of them (historical artifacts and expertise concierge) are directing the later studies for automated expertise identification. However,

this study's research setting is a mid-size company, which did not capture the collaboration model in larger or smaller companies, especially for distributed teams. Besides, due to the age of this study and the development of knowledge sharing/transferring platforms, we argue that we need to re-evaluate our expertise location practice in the organizational settings.

4.2.3 Automated Expertise Location Techniques and Systems

Since in 2000, researchers start designing approaches and systems for locating the expertise based on specific needs. Notably, these studies inherit the Historical Artifacts heuristics purposed by McDonald and Ackerman [56]. Also, these approaches aim to measure and quantify expertise of a person while mining historical artifacts.

Mining Historical Artifacts

McDonald and Ackerman 2000

Two years later, McDonald and Ackerman published an automated expert recommendation system for solving problems in software organizations called Expertise Recommender (ER) [57]. This expertise location system attempt to decrease the workload of expertise concierge and provide alternative options. They employed their expertise locating heuristics from their previous field study, and their main strategy is based on mining the historical artifacts.

There are two basic heuristics for expertise location in ER. The first one is Change History heuristic which adopted from the "Line 10 Rule" from their field study [56]. The ER system checks for the last change of the software module in the version control system, and the developer who made the last modification would be the best candidate to ask for help.

The second heuristic is inherited from the Tech Support. It enhances the routine behavior

Figure 4.5: The Expertise Request Dialog of Expertise Recommender [57]. Topic Area: Selecting the Heuristic; Filter: Strategy for Expertise Selection

of a technical support when she faced an unfamiliar problem and starting to find a similar experience or problem in the past. Therefore, ER creates a local database for querying similar problems that happened before and associated with the person who solved them.

A user can pick the basic heuristic based on which one fits their need of the current problem, and then choose the mechanism while selecting the expert (see Figure 4.5).

The two heuristics applied in this study are very inspiring and lead the future studies for expertise locations. Most future studies applied, integrated or combined these two heuristics for their design heuristics. For example, mine version control systems for changing history [60, 72] or look for the similarity of unsolved and solved problems [7, 88]. This applies two data sources, changing history from the version control system and also the database of previous tech support problems. However, this system lacks a systematic evaluation on the performance of the system.

Mockus and Herbsleb 2002

Mockus and Herbsleb designed another expertise location system called Expertise Browser (ExB) [60]. Their main purpose of this study is to locate relevant expertise for collaboration in geographically distributed development. The major contribution of their work is to start quantifying a developer's expertise on specific code module based on activities, and distinguish developer who have only brief worked on a module and who have extensively worked.

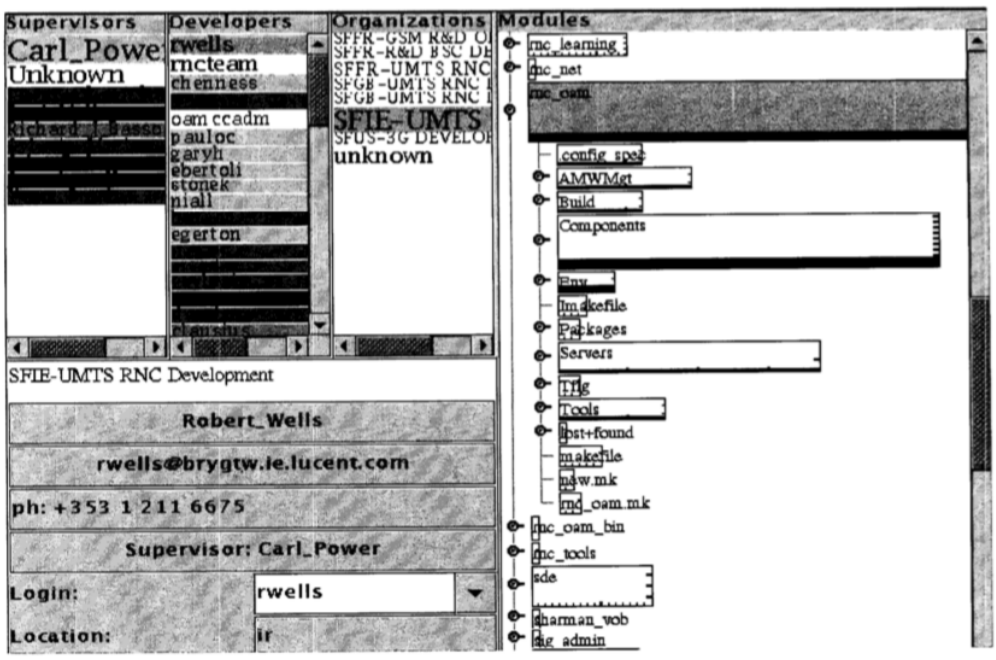


Figure 4.6: The User Interface of ExB [60]

In this study, they purposed a quantitative measurement of expertise without professional licensing. They defined the concept of *Experience Atoms* (EAs) as the smallest unit of experience, and in practice, EAs refer to "the smallest meaningful unit of such change," and the "change" is the direct result of a developer's activity on the product. By applying this concept, ExB summarizes a developer's expertise on changing files.

The user interface design of ExB is as follows (See Figure 4.6). For each software module, a developer would have different amounts of EAs which contribute to different levels of expertise on a specific model. In the right panel, the length of each bar on the module name, represents the accumulated EAs for "Robert Wells" the example developer's expertise based on his activities. Intuitively, the bigger the bar represents, the higher expertise. Finally Mockus and Herbsleb conducted a case study on distributed teams to evaluate the tool.

Noticeably, except this study first represents the expertise in software engineering practice by the form of visualization (the area of bar charts), it gives two suggestions for future studies. First, time of the change activity is also a critical factor to consider while building

the quantitative model for expertise. In this study, the time factor is discussed to help the model the difficulty of a Modification Requests. A longer time of MR interval may caused by harder request. Time is a critical factor in modeling and quantifying expertise for various aspects, and Mockus and Herbsleb were the first to include it in the context of software engineering based on our survey.

In another discussion of the study, the authors were mapping experience to expertise. Though experience is not the perfect measurement for expertise, which were not reflected by the cognitive model of expertise, their discussion provides insights on what other information can be considered in expertise measurement. Finally, they also discussed the possibility of representing a developer's expertise with visualizations.

However, this study also has its limitation. First, it only applies the single data source, which only uses the changing log from version control system to determine the expertise. Moreover, as it is a case study, it only tested ExB's usability with participants, but it lacks user feedback or evaluation on the precision (usefulness) of the experts that ExB recommended.

Expertise Concierge and Social Network Analysis

Nardi et al. 2002

Earlier in 2002, information digitization has been a trend even for our contact book. As previous study mentioned, personnel such as expertise concierge plays an essential role in locating expertise under the organization setting [56], but this type of role need to handle more information of other members in the team such as their contact information. Thus, Nardi et al. design and implement an assistant software named ContactMap [61] to not only support expertise concierges to manage the contact information, and also visualize the social network of each member.

Figure 4.7: The User Interface of ContactMap [61]

As they claimed in the paper, the major user scenarios of their tool are reminding and supporting the user. For example, reminding the user of others' identities and connections between people in their social network, particularly the contact information of these people. In addition, ContactMap also provide *awareness information* for distributed team members, such as their availability for phone calls [26].

This is an exploratory study of leveraging the social network of team members and support collaboration and communication. Moreover, they were also the pioneers in visualizing social networks In the future, they plan to address more detailed research questions such as how people use this tool (evaluation of this work)?; How to support task-specific network?; Whether to hide peripheral members of the task in ContactMap and so on. They lead the discussion on this topic.

Lin et al. 2009

As a follow-up study on the social network analysis, Lin et al. design a social network data mining tool called Smallblue [50].

In this work, they pushed the research of social network one step further by analyzing the social network and generating location of the key persons in it. They identify the key people, "Key Hubs" in the network through graph analysis such as locating structural holes. Besides the identification function, they also provide *awareness information* in their tool, such as displaying the geographical information for each team member.

Comparing with previous work, their visualization of a social network is more detailed in different perspectives and provide richer information on an individual team member. Moreover, it gives more personal information which is beyond contact information only. This tool combines the social network attribute and expertise together. Finally, it was an online web tool[1] which is maintained by IBM, though it is no longer maintained and offline now.

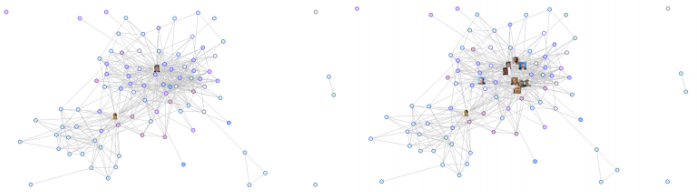
4.2.4 Open Source and Knowledge Sharing Site

Empower Open Source

In 1983, Richard Stallman launched the GNU Project with free source code sharing online. Later in 1991, Linux was released by Linus Torvalds with the license for freely modifiable source code, which significantly promoted the movement of open source software. Until 2005, the most popular decentralized version control system was created, and three years later, GitHub was launched, which fundamentally changed the development process and collaboration approach for open source projects. Later in 2010s, studies also suggests that open source contribution is also a factor to consider in the software developer hiring process. The

[1] http://smallblue.research.ibm.com

(a) Clustering of social network of experts of 'Second Life' in IBM



(b) Key bridges in this network (c) Key hubs in this network

(d) Location and network of experts of 'Second Life' in IBM United States

Figure 4.8: The User Interface of Social Network in Smallblue [50]

publicity afforded by these open source projects, dramatically promotes the development of software engineering research, particularly for empirical software engineering studies. Moreover, researchers can utilize valuable public historical data for locating expertise.

Anvik et al. 2006

As the emergence of machine learning technologies in software engineering, this study by Anvik et al. started to assigning bug reports to developers based on their historical activities. The main purpose of this study is to alleviate the burden of managing bug report repository in large open source development teams.

As the first study to locate expertise by using machine learning techniques, their approach trains the machine learning algorithm based on the historical experience of a developer. Their algorithm models the types of the bug that a developer had solved before, and predicts developers performance on an unsolved bug based on the type of the bug. Finally it generates a list of developers in order to be considered as experts on solving such a bug. To empirically validate their tool, they analyze 3426 bug reports for Eclipse.

To characterizing bug reports, they convert text in summary and description into feature vectors which can be trained by machine learning. They applied a set of heuristics to identify the expertise based on the bug resolving history. The basic four types of which they provided in the paper are:

- *If a report is resolved as FIXED, it was fixed by whoever submitted the last approved patch. The person who solved it in the last approved patch has the expertise.*
- *If a report is resolved as FIXED, it was fixed by whoever marked the report as resolved. The person who marked it as resolved has the expertise.*
- *If a report is resolved as DUPLICATE, it was resolved by whoever resolved the report of which this report is a duplicate. The person who resolved the report as duplicate has*

the expertise.

- *If a report is resolved as WORKSFORME, it was marked by the triager, and it is unknown which developer would have been assigned the report. The report is thus labeled as unclassifiable at the moment.*

In their approach, they applied a systematic method to evaluate the recommendation of the bug report resolving expertise based on the principle information retrieval technique, Precision and Recall. They regard expertise location process as a process of information retrieval, and the searching query is the description of an unsolved bug report. However, the result is not promising at that time. They have only achieved 57% and 64% of precision for two projects, and recall is not over 10% for either project. Noticeably, the ground truth for evaluation on this machine learning technique is based on cross-validation. Therefore, the developer who is identified as the expert for a bug report by this technique may not carry the expected expertise.

Though this study's result does not look so great regarding percentage numbers (especially comparing to recent bug report assignment techniques), they lead a trend of research in software engineering by applying machine learning techniques to locate expertise, and also utilize the extensive data provided in online open source repositories.

Schuler and Zimmermann 2008

In this study, Schuler and Zimmermann refined the model of expertise on code artifacts purposed in previous studies, especially the quantitative expertise model purposed by [60] in 2002 [60]. In addition to quantifying the expertise based on a developer's modification on the code artifact, they also introduce the concept of *usage expertise* which is more straightforward to collect.

The usage expertise is the knowledge that developers would also accumulate while using

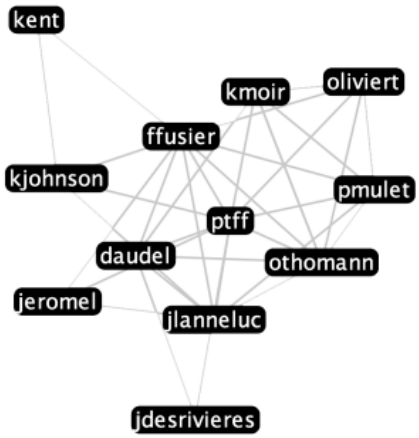


Figure 4.9: An Example of Developers Usage Expertise in Their Social Network. These Developers Mostly Use the the JDT Compiler [72]

other functionality such as calling APIs, which they did not implement. They argue that while calling (using) methods, developers might not be aware of implementation details, but these developers would gain expertise on how to use a method and when a method can be used.

Including refining the model of expertise on code artifacts, this study also mentioned the application of creating an expertise profile for a developer. However, due to the limitation of mining techniques, it only provides low-level information on a developer's expertise, and still needs human interpretation to refine high-level expertise summary. Moreover, the social network of developers is hard to extract information (see Figure 4.9).

This study did not provide an evaluation or plan for your methods. However, their contribution on the quantitative model of considering what is missing at that stage is valuable, and it leads other studies to utilize the enormous data source of version control archives fully.

Fritz et al. 2010, 2014

In their journal publication in 2014, Frtiz et al. summarized their work modeling expertise for code authorship and interaction. They purpose a model to capture the developer's expertise

on source code element, *Degree-of-knowledge*(DOK). This model utilizes the IDE to track developer's interaction with the code, and they empirically identified their results on two sites (two software development teams).

This study introduces the DOK model which represents developer's familiarity with code element. In their model, DOK is the combination of degree of authorship and degree of interest:

$$DOK = \alpha * DOA + \beta * DOI \tag{4.1}$$

Particularly, the degree of authorship data is mined from version control achieves. Defined in the paper, the degree of authorship data contains three factors (*first authorship, deliveries and acceptances*). Further, Mylyn (used to be Mylar) [45], an IDE plugin in Eclipse collects the . The degree of interest of an code element is accumulated with each interaction the developer had with it, for example, clicking or selecting a variable name rather than editing it.

For determining the weightings, α and β in the Equation 4.1, they collected the data from developers in the team, and let developer themselves to rate their knowledge about code elements as the ground truth of expertise, and then apply linear regression to decide weights in Equation 4.1.

Servant and Jones 2012

This paper introduces a technique called WhoseFault which is automated to choose expert developers to fix execution faults reflected by test cases. By applying this approach, the faulty test case can be assigned to the developer who may have the expertise to resolve the failure.

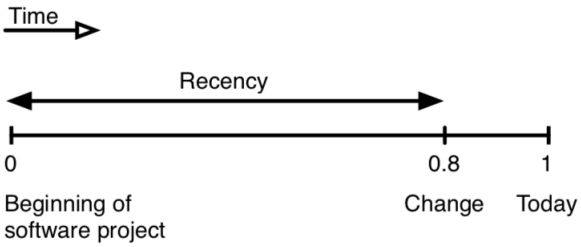


Figure 4.10: The approach of calculating *recency* based on the time of starting at the beginning of software project [74]

Second, similar to the previous work of the second author, it provides a diagnosis of where the lines of code may cause the fault.

Their approach does not only leverage the version control system to gather data but also utilizes the test cases and the execution results to complementary the expertise model. This technique extracts the history of source code with its test cases execution result to locate the fault within the source code. It applied the same technique from their previous study TARANTULA to automatically local buggy code by assigning the *suspiciousness* score to each line of the code [43]. In this study, they moved one step forward to assign execution fault to developers with expertise or responsibility based on the suspicious code.

Besides WhoseFault's implications on reducing bug reports generation as they claimed in the paper, this study also contributed to expertise location techniques, particularly on interpret and evaluate the quality of developer's previous working experience. Another critical contribution is that in their quantitative expertise model, they specifically consider the time factor and introduce the *recency* in expertise calculation see Figure 4.10. Though early study [60] has mentioned time would be a factor donates to the expertise measurement, but their emphasis was measurement on the difficulty of the task.

In their evaluation, besides comparing to other similar expertise location techniques, their ground truth of expert for fixing the fault is based on the actual developer who performed the bug fixing action later. Therefore, the developer who fixed bug is the expert. Though

this method has its limitation which raises challenges such as the bug fixer may not be the best developer with the expertise, and so on. This method improves evaluation strategies and it refers to the philosophy of evaluation approach for machine learning techniques.

Emergent Expert for Open Source

Yu et al. 2016

The pull-request model is a widely adopted method by distributed teams, especially open source ones. However, conventionally a project owner manually assign the developer to review the contribution, or the owner would review the code by herself. This manual process is time-consuming and ineffective, and it usually overburdens some core team members. As a part of expertise location research, determining several best candidates for the code reviewing task is an emergent need as open source communities getting popular in software production.

In Yu's study in 2016, they purposed a method for pull-request code reviewer assignment, which automatically recommends the expert for reviewing the source code contributed in pull-request, they identification and selection procedure grounded by previous development history. Particularly, they combine their expertise factors and common interests of developers into their expertise location model.

On GitHub, core team members for a software repository may not always be available to review the pull-request, and also because GitHub is *social coding* site, outsourcing the expertise from external reviewers is usual. Outsources also play critical roles in helping and affecting the code team members while determining whether to approve the pull-request [83]. Therefore, the interaction history of the external reviewers by comments shows their interests towards the repository, and also indirectly reflects their knowledge and expertise of the repository.

By combining this social interaction data to the experience based expertise model, they leverage the social character of the open source platforms. Noticeably, they conduct a mix method approach to evaluate their approaches. Like other recommend systems before, they adopted precision and recall to assess the performance. Moreover, they conduct a qualitative study to deeply explore the benefit of their expertise location approach which combines social interaction data. Their results suggest that technical keyword in communication traces is the most relevant factor to determine the expertise of a developer to the project, and for a project has most reviews are insiders, a developer who becomes a *dominant reviewer* usually leaves her communication traces in most of the pull-requests. For this type of collaboration mode, one repository only has a few dominant reviewers.

Costa et al., 2016

Besides the pull-request model, merging is another important collaboration practice, as parallel development is beneficial to manage time to release the product either in open source communities or commercial development. However, merging is not an easy task to perform as it requires complex expertise in resolving conflicts [22].

Current techniques and tools only detect straightforward and direct conflict such as textual conflict. However, these tools are not able to realize complex cases such as unseen dependency modifications [76]. Therefore, it is very typical to assign a developer to manually resolve the conflict or confirm if the merge is free from potential conflicts. However, it is not clear to find the appropriate developer or locate a few experts to perform the merge action.

In their study, they purposed an automated approach called TIPMerge, a novel tool which identifies a list of developers as the best candidates to perform the merge action. Similar to other expertise location systems and approaches, it determines the expertise for performing the merge based on developer's previous experience with the branch and the project, particularly the interactions with key files for the merge and their dependencies (See Figure

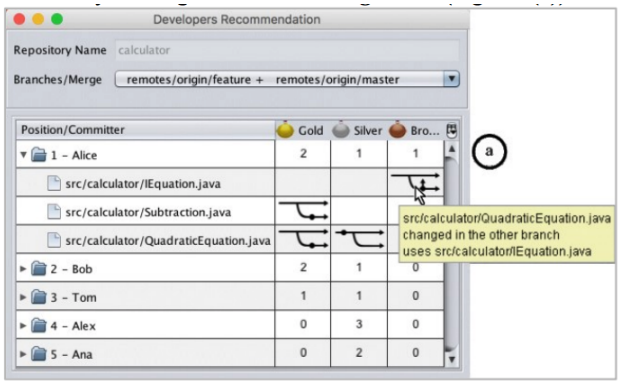


Figure 4.11: User Interface of Showing Rankings of Developers for Performing Merge in TipMerge [22]

4.11).

Finally, they evaluate their system based on a mix-method approach similar to [93]. They not only quantitatively compared their results with conventional methods, but also conduct a qualitative study by interviewing team members for two projects. As they summarized, there are couple reasons other than previous experience to decide a merger, such as, "Line 10 Rule" for merge (the developer who made the last commit would help the merge); knowledge on the trick part of the code artifact; personal preference.

Knowledge Sharing Sites

Effective knowledge sharing platform has been emergent since late 2000s (Stackoverflow launched since 2008). The free knowledge crowdsourcing practice has changed the way how developers get basic knowledge on programming questions. It was a brilliant idea for Stackoverflow to organize the documents for each programming language and API, but the company has to end the project due to its enormous effort in management and low profit for return[2]. However, a complete documentation or code examples is necessary and affordable for private knowledge resources such as knowledge sharing repository for private

[2]Stackoverflow Meta, Sunsetting Documentation: https://meta.stackoverflow.com/questions/354217/sunsetting-documentation

companies and commercial project, for example, Software Engineer from Facebook, Satish Chandra introduced the internal code repository of Facebook on 2018 ISR research Forum at University of California, Irvine[3]. Properly utilize the knowledge sharing site to locate expertise is critical but the current research has been struggling with the extracting relevant information partially due to the ambiguity of natural language. Research on knowledge sharing sites is still in the beginning and building infrastructure phase.

Hanrahan et al. 2012

It is intuitive that a person with high expertise on the domain could answer hard related questions in software engineering, and novices in the domain could not. However, the difficulty of questions is hard to model. In this study, Hanrahan et al. employ a straightforward method to model the difficulty of a question on Stackoverflow by calculating the time duration between the problem is posted and the accepted answer is posted.

It is not a brand new idea to model the difficulties and expertise. Earlier in the study of [60], they have already discussed the time issue of a task in software engineering, such as a hard request or bug would take more time to complete since it was issued. However, they did not include the time factor in their quantitative expertise model. In this study, the authors only consider the solving time as a factor, but they missed the other factors such as the popularity of the topic. However, they provide an initial attempt to model the expertise in questions through time factor.

Xu et al. 2016

Due to the ambiguity of natural language, and different representation may refer to the same semantic meaning, questions on Stackoverflow may be repeated or high related to each other. Removing the redundancy and simplifying the variations are critical for modeling the users expertise and avoid information overload for knowledge providers and receivers on knowledge

[3]https://isr.uci.edu/content/2018-isr-research-forum

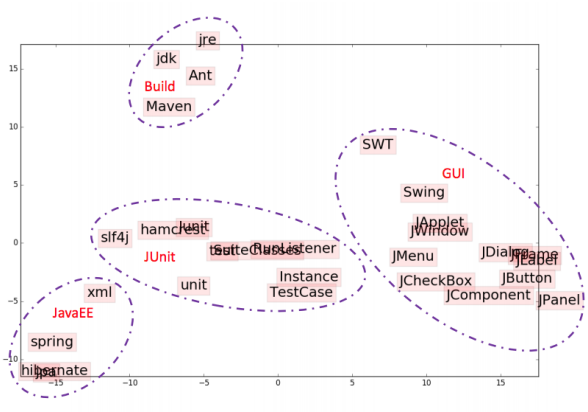


Figure 4.12: Word Representations Embedding Examples [88]

sharing sites. Unfortunately, Stackoverflow is not able to manage its enormous knowledge base like Wikipedia[4] which links keywords, i.e., knowledge unit, to its own explanations. However, the failure of Stackoverflow Documentation suggests the complicity of programming knowledge unit is not affordable for human labor to classify.

Therefore, Xu et al. employ the conventional neural network to model the knowledge unit in questions and answers for expertise. Through the techniques and approach described in their study, they can cluster different word representations into different high-level categories 4.12. Besides, noticeably they also use precision and recall as the primary evaluation metrics for their approach.

Based on their study, it is possible to manage the knowledge unit, defined in their study, and link them as Wikipedia. It enhances the usability of the site and also easier to mine a knowledge unit related expertise for developers which is at a higher level than "tag" in Stackoverflow.

4.2.5 Multi-Platform Data Aggregation

Sarma et al. 2016

[4]https://www.wikipedia.org/

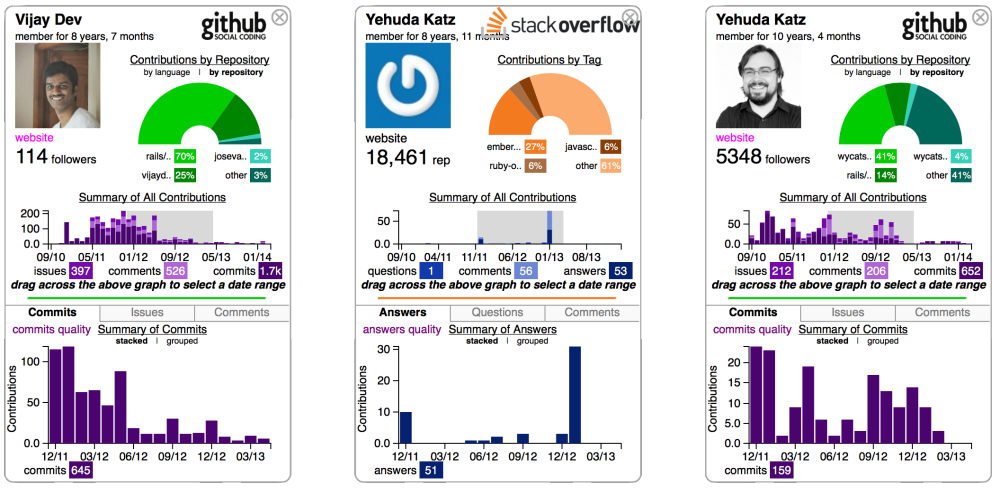


Figure 4.13: The Card Based User Interface for Comparing Expertise in the Hiring Procedure [70]

As the emerging of online knowledge sharing platform, e.g., Stackoveflow and open source platform e.g., GitHub, it is necessary to monitor and aggregate a developer's activities all over the internet for precisely measuring her expertise in software engineering for both "soft" social skills and also technical skills.

Therefore, Sarma et al. design and implement a visualization tool, Visual Resume to include a developer's activities over the internet. Notably, other than the amount of the contribution for a developer over his career (bar charts in the middle of each card in Figure 4.13), it also provides quality indicators of their contribution such as whether a pull-request is accepted and whether their commit passed the test cases provided in the repository. Besides, it employs a card-based user interface to allow easy comparison between developers, and mainly they design this tool to extract expertise from historical contributions for hiring purpose.

As for evaluation, they conduct the interview based usability test with management level of personnel from industry, and let participants either use the web portals of developer's profile or use Visual Resume to evaluate job candidates based on job description, and then select best candidates for the job. Particularly, other than finding Visual Resume is help-

ing participants in making their decisions, participants also mentioned that the qualitative interview phase is still necessary as a part of hiring new employees.

4.3 Findings Summary

In this section, we summarize the major findings from our survey review. First, we present a historical view of expertise location systems and approaches in the development of software engineering. Further, we argue that there is some limitations of the current trend of expertise location systems, especially on their dataset to determine the expertise and evaluation methods for the system. Finally, we provide some suggestions based on the current research status.

4.3.1 Evolution of Automated Techniques: From Locating Expertise of "Playing Chess" to "Moving Pawn"

In this study, we reviewed the studies of expertise location in a historical perspective. We summarize the development of expertise location studies in the figure (see Figure 4.14).

Early expertise research, researchers were discovering the characteristics of experts and masters, and they found experts perform better in their expertise domain mainly due to their higher information processing abilities, and this conclusion has been confirmed by a few studies in software engineering which found senior expert programmers can also process information of code faster and more precise.

Since the field study by McDonald and Ackerman in 1998 [56], they set up the heuristics for expertise location systems and approaches later, mostly mining the historical artifacts or through expertise concierge. By studying our paper repository, we found automated

expertise location techniques inherit and employ these two heuristics when manually locating experts. However, to the best of our knowledge, the research of expertise concierge has mostly vanished since IBM SMALLBLUE [50]. On the other hand, expertise research that mines historical artifacts goes to a direction of similar granularity, and automated expertise location research has a trend of going to the micro level of tasks. For example, recent research [22, 93] focus on specific micro tasks in open source software engineering such as merging different branches. We found that this trend is necessary for performing certain low-level task ("*moving pawn*"). However, when stakeholders need to perform high-level task such as locating the best candidate for training, or hiring new employees among candidates ("*playing chess*"), low-level expertise indicators such as expertise in calling APIs is usually not intuitive to help decide the best candidates for an engineer job.

4.3.2 Expertise and Time

Only very few studies consider that expertise is not a static attribute of developers. [70] found the fact from their user study that practitioners in hiring committees of programming jobs decide a job candidate's expertise for programming task and their availability partial based on their recent activities [70]. Servant and Jones also consider recency as factor in their statistical model of measuring expertise for dealing with fault, i.e., with the same amount of contribution if a developer submitted the code more recently then this developer has the higher expertise in related issue. Early studies by [60] also mentioned the time issue in measuring expertise, but they intend to use the time as a measure for quantifying the difficulties and expertise required to perform the task which is under another topic. We argue that developers' expertise is continueously accumulating and diminishing. It is non-trivial to consider the time factor when measuring expertise.

4.3.3 Evaluation of Expertise Location Approaches and Systems

In this study, we found a variety of evaluation methods for expertise location approaches, but generally, previous researchers were struggling in evaluating the expertise location research and especially building the ground truth of their studies.

Earlier expertise location research usually conduct a usability test with practitioners [60], or even without an evaluation phase or plan [57, 61]. Later, *Precision and Recall* becomes one of the most popular evaluation approaches to assess the effectiveness of an expertise location tool [7, 88, 93]. However, it is still hard for researchers to build ground truth for the result of expertise location, and cross-validation is the general solution. Mainly, slicing the history at a certain point of time, uses the data before to create evidence data for locating expertise, and then employ later data (e.g., the developer who actually fixed the bug) as ground truth in evaluation. Though there are more factors lying between the best candidate for the task, and the person who actually performs the task, such as availability of the expert. Besides, the complex environment and significant turnover of developer in open source entail more reasons of the person who performs the task. However, employing the historical data is the most generic approach for building ground truth of expertise.

Due to the limitation in the evaluation of these studies, recent studies adopt a mix-method approach to build and validate the ground truth for evaluating expertise location techniques [22, 88, 93]. With a detailed qualitative study, these studies can reveal the expertise sharing network inside actual teams and identify the experts among them, but since the cost of these studies, researchers are not able to provide such detailed qualitative study for various environment.

4.3.4 Call for a Field Study

The last field study for locating expertise in the domain of software engineering is conducted at 1998 [56], when software organizations were all private without open source communities, and also developers activities and behaviors records were stored locally inside the company. Therefore the earlier expertise location practice focus on the private data sources, including organization staff who played as key roles like expertise concierges.

However, it has been 20 years since the early field study, and both the supporting techniques and collaboration practice have changed. The emerging techniques such as machine learning techniques which support software engineering activities analysis, and natural language processing techniques which classify textual description, are significantly improving the experience of software development. Recent collaboration practice such as widely using the DVCS in parallel programming for saving time to the market. Moreover, the internal and public knowledge sharing site is also changing the strategy of developers while transferring expertise and solving problems.

We argue that there is a need of questionnaire surveys, observations, or field studies to explore the expertise location approaches and heuristics in the modern software engineering practice. Moreover, there is a need to explore the role of open source communities, knowledge sharing sites, professional social network and automated expertise location tools in the modern practice. The future study shall be conducted for different purposes of expertise location studies such as hiring, training, collaboration and performing task.

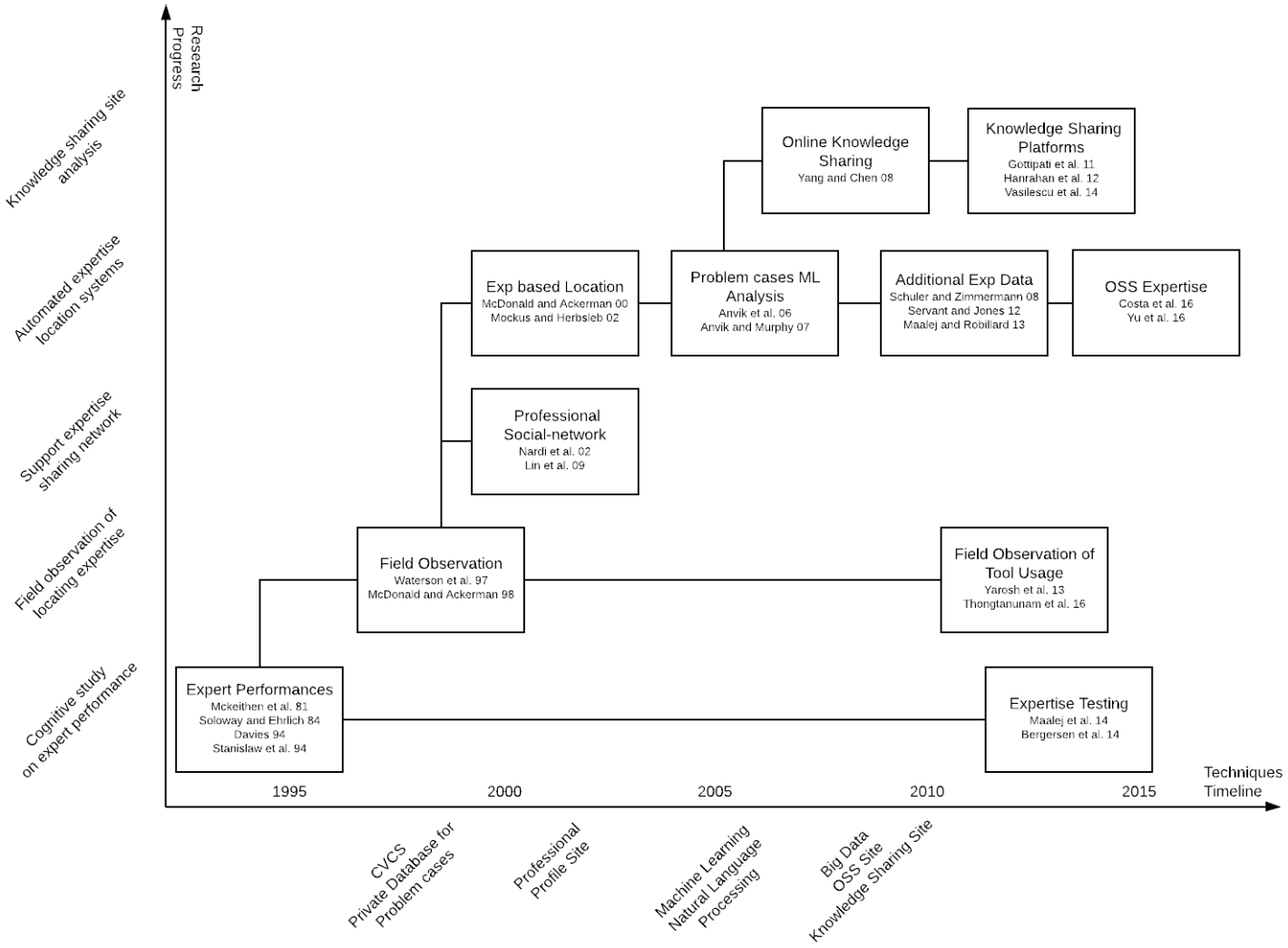


Figure 4.14: The Development History of Research in Expertise Location Systems and Approaches

Chapter 5

Discussion

In this section, we provide several points of discussion for this study, including identifying the disconnection between discussions on the potential improvement of the expertise model, and other data sources to include. We also discuss the critical problem about the gap between early cognitive research on expert performance and current expertise location studies in software engineering. Further, We provide our suggestions to bridge this gap.

5.1 "How Well Do They Perform" v.s. "What Have They Done"

By reviewing our repository, we found that the quantitative expertise model has been evolving continuously, which adds new data sources, such as adding interaction data [35], and including API usage data [72] into their expertise measurement model. However, these expertise models still have failed to measure the cognitive characteristics of experts, which were identified by early cognitive studies [37, 58, 77]. Based on these previous researches, experts process, i.e., receive and recall, their domain information faster and more precisely. This su-

perior ability grants them with higher proficiency in their domains. The excellence of their information processing ability helps them to realize the best solution and handle unexpected situation [30] automatically. Conclusion from these cognitive studies is evidenced by empirical studies which monitoring the experts' performance during the task (monitoring how well expert perform). However, the heavy experience-based expertise models (measuring what have expert done) fail to reflect developers ability in processing information in the field of software engineering. We argue that there is a **disconnect** between early cognitive studies on expert performance and the software engineering approaches and systems for locating expertise, and over-rely on the experience-based expertise model may bias the assessment of expertise. There is one study that aims at creating measurements for evaluating the expertise level of a developer [10], but to the best of our knowledge, we are not able to find a location approach that assesses the information processing ability. Moreover, the automated location systems mainly employ data of subject's previous experience.

5.2 Community Feedback v.s. Automated Expertise Location Techniques

Employing human judgment to assess the expertise of a developer is common practice in the industry while locating expertise [56]. For example, the approaches of "*everyday expertise*" and "*expertise concierge*" are employing human judgment to determine expertise [56]. This approach relies a human, such as a peer, to assess the expertise of developers especially considering acknowledgement from dignitaries in the community. There is a previous study that also includes feedback data into the expertise database and creates individual expertise profile partially based on the it [67]. As the emergence of business and employment-oriented service sites such as LinkedIn, it certifies the users' expertise based on the *endorsement* from their peer in their connection circle and other LinkedIn users [2], and GitHub has

Figure 5.1: The Endorsement System in LinkedIn. A list of User Profile Pictures on the Right of Each "Skills & Expertise", and Number on the Left Indicates How Many Connections in Their Social Network Have Certified This Item of Skill [1].

popular repositories and third-party sites which rank followers[1] and stars[2] received from the community.

However, this type of evaluation of expertise can be very easily manipulated, similar to Twitter where users can purchase followers by number. In a previous study [55], user study participants have already mentioned that popularity is an indicator but not always enough to evaluate the quality of work. We argue the *endorsement* on LinkedIn is also an indicator based on popularity but specific about the skills and expertise of a developer.

In the future study, we purpose to carefully integrate the user rating and popularity based expertise evaluation with the automated techniques which mostly retrieve the activity history

[1]Following People, https://help.github.com/articles/following-people/
[2]About Stars, https://help.github.com/articles/about-stars/

of a developer. However, we need to find a balance between employing user feedback and the historical data, but avoid over-reliance on either single data source.

5.3 Paradox of Expertise

The ultimate goal of locating expert is to find someone who is qualified to perform a job or task. It is non-trivial to determine the high-level goal of locating expertise firstly. However, even if experts have experienced adequate training in the past, when it comes to a specific task that requires to share expertise, such as training and coaching new employers, the person with highest expertise may not be the best candidate for performing these tasks due to the paradox of expertise [27], since their own experience and capacity bias them [52].

Experts think differently from novices. Their above-average performance in information processing often result from a pre-determined and routine mechanism. This mechanism is highly effective but also restricts the flexibility and control while expressing and articulating their specialties [27]. For example, one of the best basketball players in the history, Michael Jordan, has never been a head coach of any basketball teams. It is determined by his extremely high motion expertise in basketball. Therefore, performing skillful layups over a defensive player for Michael is an easy routine which has his fixed mechanism. Michael is capable of reacting to and avoiding blocks perfectly based on the environmental information such as defender's height and distance to him, but his mechanism of avoiding defense also restricts his expression for his drills. [79].

In the study of Dror [27], the author claims that one of the major reasons causing this paradox is the heavy amount of training that expert has received. Once they formed an automated routine, their expertise is not easily accessible. However, spending effort for accessing this effort degrades experts' performance [34]. Therefore, expertise location studies

need to consider this paradox of expert, and while locating expertise for expertise sharing tasks, novices may help experts to avoid this paradox. For example, Arthur Andersen LLP[3] allocates their new hires called "green beans", into expert teams [5]. These "green beans" ask basic questions such as definition of terminologies, which helps the experts access their expertise when green beans asking questions [44].

5.4 Guiding Entry-level Novices

Conventional expertise location techniques are usually employed in recommending people for performing specific software engineering task, and also in hiring, training and allocating software talents [10]. As open source software is playing an essential role in the software industry, and due to its voluntary attribute [75], it is non-trivial to guide and encourage more participants to the community. As previous research suggested, newcomers of open source projects are facing various heavy barriers to entering the community [81].

Steinmacher et al. has identified five main categories of these barriers [81]. The major categories of these barriers are:

- *Technical Hurdles*: difficulties in understanding the code or setting up the workspace and environment.
- *Documentation*: difficulties in understanding the project.
- *Social interactions*: communication issues with the previous contributors of the project.
- *Newcomers previous knowledge and expertise*: similar to technical hurdles, but more focusing on the newcomer's technical expertise such as experience in software engineering practice.

[3]Arthur Andersen LLP was one of the largest accounting firm in the US

- *Starting point*: difficulties in finding a mentor or entry-level task to start.

Except the difficulty in social interaction, other barriers can be addressed or alleviated by sophisticated expertise location systems, particularly in finding a mentor with required expertise to guide or help the novice to perform the technical task [9]. Further, to support newcomers to find the available expertise in the project and then enter the community for contribution, it also shows the non-trivial requirement of the expertise location systems which is including the constraints of the others such as their availability.

5.5 Limitations

In this study, we reviewed the expertise location approaches and systems in Software Engineering and Computer Supported Collaborative Work based on our research questions. However, during our research procedure, we found early cognitive studies are not only focused on and limited in the field of Software Engineering. Though there are early studies about the relation between the information processing ability and programming expertise in the software engineering practice, we are not able to locate any studies that aimed to systematically build theories for measuring the expertise level for software engineering personnel. Due to the scope of this study, we only focus the papers in the field of Software Engineering and Computer Supported Collaborative Work; we may miss the progress in cognitive studies which analyze expert behaviors and characteristics. In the future study, we also intend to explore studies from other research subjects other than these two we have already reviewed in this study, particularly for field in management, organizational and cognitive science. Besides, we find related literature is not limited in the form of paper publications. There are lots of valuable resources in books [30, 77], and we intend to include these resources as well.

Chapter 6

Conclusion

In this study, we reviewed 48 primary studies about expertise location approaches and systems in the field of Software Engineering and Computer Supported Collaborative Work. We found that the conclusions in software engineering are similar to early cognitive research on expert performance. Moreover, we found several major categories of expertise location approaches and systems measuring the expertise level based on the previous experience of developers which results a disconnect between early cognitive expertise research and automated location techniques in Software Engineering practice. Further, we found the granularity of model for measuring expertise in these location techniques trend to be more focused on micro low level tasks over the years, and it is challenging to evaluate the measurement of expertise and location approach in the real-world context. Finally, we conclude our findings, and provide a series of discussions on future directions for expertise location studies in Software Engineering.

Bibliography

[1] "Linkedin endorsements: Why they may be hurting you," Apr 2013. [Online]. Available: https://sunsender.wordpress.com/2013/04/22/linkedin-endorsements-why-they-may-be-hurting-you/

[2] "Linkedin help." [Online]. Available: https://www.linkedin.com/help/linkedin/answer/31888/skill-endorsements-overview?lang=en

[3] M. S. Ackerman, J. Dachtera, V. Pipek, and V. Wulf, "Sharing knowledge and expertise: The cscw view of knowledge management," *Computer Supported Cooperative Work (CSCW)*, vol. 22, no. 4-6, pp. 531–573, 2013.

[4] T. J. Allen, "Managing the flow of technology: Technology transfer and the dissemination of technological information within the r & d organization(book)," *Research supported by the National Science Foundation. Cambridge, Mass., MIT Press, 1977. 329 p*, 1977.

[5] K. B. and and I. J. Dugan, "Arthur andersen's fall from grace is a sad tale of greed and miscues," Jun 2002. [Online]. Available: https://www.wsj.com/articles/SB1023409436545200

[6] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug reports," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007, p. 2.

[7] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 361–370. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134336

[8] R. Bednarik, "Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations," *International Journal of Human-Computer Studies*, vol. 70, no. 2, pp. 143–155, 2012.

[9] A. Begel and B. Simon, "Novice software developers, all over again," in *Proceedings of the Fourth international Workshop on Computing Education Research*. ACM, 2008, pp. 3–14.

[10] G. R. Bergersen, D. I. Sjøberg, and T. Dybå, "Construction and validation of an instrument for measuring programming skill," *IEEE Transactions on Software Engineering*, vol. 40, no. 12, pp. 1163–1184, 2014.

[11] M. Bilalić, *The Neuroscience of Expertise*, ser. Cambridge Fundamentals of Neur. Cambridge University Press, 2017.

[12] D. Bobrow and A. Collins, *Representation and Understanding: Studies in Cognitive Science*, ser. Language, Thought and Culture Series. Elsevier Science & Technology Books, 1975. [Online]. Available: https://books.google.com/books?id=Pd9OAAAAMAAJ

[13] G. Bohlander and S. Snell, *Managing Human Resources*, ser. Available Titles Aplia Series. South-Western Cengage Learning, 2010. [Online]. Available: https://books.google.com/books?id=J1MgADAhyr8C

[14] F. P. J. Brooks, "No silver bullet essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, April 1987.

[15] F. Brooks, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Pearson Education, 1995. [Online]. Available: https://books.google.com/books?id=Yq35BY5Fk3gC

[16] J. Chan, S. Dang, and S. P. Dow, "Improving crowd innovation with expert facilitation," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 2016, pp. 1223–1235.

[17] W. G. Chase and H. A. Simon, "Perception in chess," *Cognitive psychology*, vol. 4, no. 1, pp. 55–81, 1973.

[18] Y. Chen, S. Oney, and W. S. Lasecki, "Towards providing on-demand expert support for software developers," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 3192–3203.

[19] A. Chevalier and M. Y. Ivory, "Web site designs: Influences of designer's expertise and design constraints," *International Journal of Human-Computer Studies*, vol. 58, no. 1, pp. 57–87, 2003.

[20] M. T. Chi, "Two approaches to the study of experts characteristics."

[21] N. J. Cohen and L. R. Squire, "Preserved learning and retention of pattern-analyzing skill in amnesia: Dissociation of knowing how and knowing that," *Science*, vol. 210, no. 4466, pp. 207–210, 1980.

[22] C. Costa, J. Figueiredo, L. Murta, and A. Sarma, "Tipmerge: recommending experts for integrating changes across branches," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 523–534.

[23] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1277–1286. [Online]. Available: http://doi.acm.org/10.1145/2145204.2145396

[24] S. P. Davies, "Knowledge restructuring and the acquisition of programming expertise," *International Journal of Human-Computer Studies*, vol. 40, no. 4, pp. 703–726, 1994.

[25] A. de Groot, *Thought and Choice in Chess*, ser. Amsterdam Academic Archive Series. Amsterdam University Press, 1965.

[26] P. Dourish and V. Bellotti, "Awareness and coordination in shared workspaces," in *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. ACM, 1992, pp. 107–114.

[27] I. E. Dror, "The paradox of human expertise: why experts get it wrong," *The paradoxical brain*, vol. 177, 2011.

[28] Y. Dudai, *Memory From a to Z: Keywords, Concepts, and Beyond*. Oxford University Press, 2004.

[29] K. Ehrlich and D. Cash, "Turning information into knowledge: information finding as a collaborative activity," in *Proceedings of Digital Libraries' 94*, 1994, pp. 119–125.

[30] K. Ericsson, N. Charness, P. Feltovich, and R. Hoffman, *The Cambridge Handbook of Expertise and Expert Performance*, ser. Cambridge Handbooks in Psychology. Cambridge University Press, 2006.

[31] Y. Fathy, P. Barnaghi, and R. Tafazolli, "Large-scale indexing, discovery, and ranking for the internet of things (iot)," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 29:1–29:53, Mar. 2018. [Online]. Available: http://doi.acm.org/10.1145/3154525

[32] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[33] G. Fischer, "Desert island: software engineeringa human activity," *Automated Software Engineering*, vol. 10, no. 2, pp. 233–237, 2003.

[34] K. E. Flegal and M. C. Anderson, "Overthinking skilled motor performance: Or why those who teach cant do," *Psychonomic Bulletin & Review*, vol. 15, no. 5, pp. 927–932, 2008.

[35] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 385–394.

[36] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: Modeling a developer's knowledge of code," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 2, p. 14, 2014.

[37] F. Gobet and H. A. Simon, "Recall of random and distorted chess positions: Implications for the theory of expertise," *Memory & cognition*, vol. 24, no. 4, pp. 493–503, 1996.

[38] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 323–332.

[39] G. J. Greene and B. Fischer, "Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 2016, pp. 804–809.

[40] D. Guralnik, *Webster's New World Dictionary*. Simon and Schuster, 1983.

[41] C. Hannebauer, M. Patalas, S. Stünkelt, and V. Gruhn, "Automatically recommending code reviewers based on their expertise: An empirical comparison," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 2016, pp. 99–110.

[42] B. V. Hanrahan, G. Convertino, and L. Nelson, "Modeling problem difficulty and expertise in stackoverflow," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion*. ACM, 2012, pp. 91–94.

[43] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th international conference on Software engineering*. ACM, 2002, pp. 467–477.

[44] P. T. Kelly and C. E. Earley, "Leadership and organizational culture: Lessons learned from arthur andersen," *Accounting and the public interest*, vol. 9, no. 1, pp. 129–147, 2009.

[45] M. Kersten and G. C. Murphy, "Mylar: a degree-of-interest model for ides," in *Proceedings of the 4th international conference on Aspect-oriented software development*. ACM, 2005, pp. 159–168.

[46] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Information and Software Technology*, vol. 55, no. 12, pp. 2049 – 2075, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584913001560

[47] M. Kläs, F. Elberzhager, J. Münch, K. Hartjes, and O. von Graevemeyer, "Transparent combination of expert and measurement data for defect prediction: an industrial case study," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 119–128.

[48] J. Koenemann and S. P. Robertson, "Expert problem solving strategies for program comprehension," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1991, pp. 125–130.

[49] B. Ley, T. Ludwig, V. Pipek, D. Randall, C. Reuter, and T. Wiedenhoefer, "Information and expertise sharing in inter-organizational crisis management," *Computer Supported Cooperative Work (CSCW)*, vol. 23, no. 4-6, pp. 347–387, 2014.

[50] C.-Y. Lin, N. Cao, S. X. Liu, S. Papadimitriou, J. Sun, and X. Yan, "Smallblue: Social network analysis for expertise search and collective intelligence," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 2009, pp. 1483–1486.

[51] Z. Lin, Y. Zou, J. Zhao, and B. Xie, "Improving software text retrieval using conceptual knowledge in source code," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017, pp. 123–134.

[52] B. S. Liu, *The expertise paradox: Examining the role of different aspects of expertise in biased evaluation of scientific information*. University of California, Irvine, 2013.

[53] W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.

[54] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the comprehension of program comprehension," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, p. 31, 2014.

[55] J. Marlow and L. Dabbish, "Activity traces and signals in software developer recruitment and hiring," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 145–156.

[56] D. W. McDonald and M. S. Ackerman, "Just talk to me: a field study of expertise location," in *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. ACM, 1998, pp. 315–324.

[57] ——, "Expertise recommender: a flexible recommendation system and architecture," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 2000, pp. 231–240.

[58] K. B. McKeithen, J. S. Reitman, H. H. Rueter, and S. C. Hirtle, "Knowledge organization and skill differences in computer programmers," *Cognitive Psychology*, vol. 13, no. 3, pp. 307 – 325, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0010028581900128

[59] S. Minto and G. C. Murphy, "Recommending emergent teams," in *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*. IEEE, 2007, pp. 5–5.

[60] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th international conference on software engineering*. ACM, 2002, pp. 503–512.

[61] B. A. Nardi, S. Whittaker, E. Isaacs, M. Creech, J. Johnson, and J. Hainsworth, "Integrating communication and information through contactmap," *Communications of the ACM*, vol. 45, no. 4, pp. 89–95, 2002.

[62] G. M. Olson and J. S. Olson, "Distance matters," *Human–computer interaction*, vol. 15, no. 2-3, pp. 139–178, 2000.

[63] A. Paepcke, "Information needs in technical work settings and their implications for the design of computer tools," *Computer Supported Cooperative Work (CSCW)*, vol. 5, no. 1, pp. 63–92, 1996.

[64] J. Pinto and E. Soloway, "Providing the requisite knowledge via software documentation," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1988, pp. 257–261.

[65] V. Pipek, V. Wulf, and A. Johri, "Bridging artifacts and actors: Expertise sharing in organizational ecosystems," *Computer Supported Cooperative Work (CSCW)*, vol. 21, no. 2-3, pp. 261–282, 2012.

[66] M. M. Rahman, C. K. Roy, and J. A. Collins, "Correct: code reviewer recommendation in github based on cross-project and technology experience," in *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*. IEEE, 2016, pp. 222–231.

[67] T. Reichling, M. Veith, and V. Wulf, "Expert recommender: Designing for a network organization," *Computer Supported Cooperative Work (CSCW)*, vol. 16, no. 4, pp. 431–465, Oct 2007. [Online]. Available: https://doi.org/10.1007/s10606-007-9055-2

[68] R. Revlin, *Cognition: Theory and practice*. Macmillan, 2012.

[69] H. B. Richman, J. J. Staszewski, and H. A. Simon, "Simulation of expert memory using epam iv." *Psychological Review*, vol. 102, no. 2, p. 305, 1995.

[70] A. Sarma, X. Chen, S. Kuttal, L. Dabbish, and Z. Wang, "Hiring in the global stage: Profiles of online contributions," in *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, Aug 2016, pp. 1–10.

[71] R. Saxena and N. Pedanekar, "I know what you coded last summer: Mining candidate expertise from github repositories," in *Companion of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. ACM, 2017, pp. 299–302.

[72] D. Schuler and T. Zimmermann, "Mining usage expertise from version archives," in *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 2008, pp. 121–124.

[73] F. Servant and J. A. Jones, "History slicing," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 452–455. [Online]. Available: http://dx.doi.org/10.1109/ASE.2011.6100097

[74] ——, "Whosefault: automatic developer-to-fault assignment through fault localization," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 36–46.

[75] S. K. Shah, "Motivation, governance, and the viability of hybrid forms in open source software development," *Management science*, vol. 52, no. 7, pp. 1000–1014, 2006.

[76] E. Shihab, C. Bird, and T. Zimmermann, "The effect of branching strategies on software quality," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2012, pp. 301–310.

[77] H. A. Simon, *The Sciences of the Artificial (3rd Ed.)*. Cambridge, MA, USA: MIT Press, 1996.

[78] E. Soloway and K. Ehrlich, "Empirical studies of programming knowledge," *IEEE Transactions on software engineering*, no. 5, pp. 595–609, 1984.

[79] N. Staff and NBA.com, "Legends profile: Michael jordan," Aug 2017. [Online]. Available: http://www.nba.com/history/legends/profiles/michael-jordan

[80] H. Stanislaw, B. Hesketh, S. Kanavaros, T. Hesketh, and K. Robinson, "A note on the quantification of computer programming skill," *International journal of human-computer studies*, vol. 41, no. 3, pp. 351–362, 1994.

[81] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Information and Software Technology*, vol. 59, pp. 67 – 85, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914002390

[82] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the 38th international conference on software engineering*. ACM, 2016, pp. 1039–1050.

[83] J. Tsay, L. Dabbish, and J. Herbsleb, "Let's talk about it: evaluating contributions through discussion in github," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. ACM, 2014, pp. 144–154.

[84] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social q&a sites are changing knowledge sharing in open source software communities," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 342–354.

[85] Z. Wang, H. Sun, Y. Fu, and L. Ye, "Recommending crowdsourced software developers in consideration of skill improvement," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017, pp. 717–722.

[86] P. E. Waterson, C. W. Clegg, and C. M. Axtell, "The dynamics of work organization, knowledge and technology during software development," *International Journal of Human-Computer Studies*, vol. 46, no. 1, pp. 79–101, 1997.

[87] E. J. Weyuker, "Testing component-based software: A cautionary tale," *IEEE software*, vol. 15, no. 5, pp. 54–59, 1998.

[88] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 51–62.

[89] S. J. Yang and I. Y. Chen, "A social network-based system for supporting interactive collaboration in knowledge sharing over peer-to-peer network," *International Journal of Human-Computer Studies*, vol. 66, no. 1, pp. 36–50, 2008.

[90] S. Yarosh, T. Matthews, M. Zhou, and K. Ehrlich, "I need someone to help!: a taxonomy of helper-finding activities in the enterprise," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 1375–1386.

[91] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 689–699.

[92] Y. Ye and G. Fischer, "Supporting reuse by delivering task-relevant and personalized information," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 513–523. [Online]. Available: http://doi.acm.org/10.1145/581339.581402

[93] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Technology*, vol. 74, pp. 204–218, 2016.