FEDERAL UNIVERSITY OF CEARÁ

CENTER OF SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

POST-GRADUATION PROGRAM IN COMPUTER SCIENCE

MASTER'S DEGREE IN COMPUTER SCIENCE

GABRIEL PASSOS URANO DE CARVALHO

ADAPTING THE RETURNERS AND EXPLORERS DICHOTOMY TO UNDERSTAND SOURCE CODE KNOWLEDGE DISTRIBUTION

FORTALEZA

2025

GABRIEL PASSOS URANO DE CARVALHO

ADAPTING THE RETURNERS AND EXPLORERS DICHOTOMY TO UNDERSTAND SOURCE CODE KNOWLEDGE DISTRIBUTION

Dissertation submitted to the Post-Graduation Program in Computer Science of the Center of Sciences of the Federal University of Ceará, as a partial requirement for obtaining the title of Master in Computer Science. Concentration Area: Software Engineering

Advisor: Prof. Dr. Lincoln Souza Rocha

Co-advisor: Profa. Dr. Emanuele Marques Rodrigues Santos

FORTALEZA

2025

GABRIEL PASSOS URANO DE CARVALHO

ADAPTING THE RETURNERS AND EXPLORERS DICHOTOMY TO UNDERSTAND SOURCE CODE KNOWLEDGE DISTRIBUTION

Dissertation submitted to the Post-Graduation Program in Computer Science of the Center of Sciences of the Federal University of Ceará, as a partial requirement for obtaining the title of Master in Computer Science. Concentration Area: Software Engineering

Approved on: 29th August 2025

EXAMINATION BOARD

Prof. Dr. Lincoln Souza Rocha (Advisor)
Federal University of Ceará (UFC)

Profa. Dr. Emanuele Marques Rodrigues Santos (Co-advisor)
Federal University of Ceará (UFC)

Prof. Dr. Guilherme Amaral Avelino
Federal University of Piauí (UFPI)

Prof. Dr. Igor Fabio Steinmacher
Northern Arizona University (NAU)

Dedicated to my family, my partner, and my friends, whose support, encouragement, and companionship made this journey possible.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Prof. Dr. Lincoln Rocha for his dedicated and precise supervision throughout this dissertation. His guidance was fundamental to the intellectual development and maturation of this work.

I am also deeply grateful to Prof. Dr. Emanuele Santos, my co-advisor, for her generosity, insightful discussions, constant support, and valuable contributions to this research.

My sincere thanks go to Prof. Dr. Ronaldo Menezes, from the University of Exeter, for his careful review of the manuscripts and for sharing his expertise on the original Returners and Explorers framework that inspired this research.

I am deeply grateful to my parents and siblings for providing me with a strong foundation of education and values, as well as for their patience, encouragement, and understanding throughout the completion of this work.

I would also like to thank my partner, Laura, for her constant support, patience, and encouragement during the most challenging stages of this journey. Her presence was essential throughout the development of this dissertation.

I also thank Thiago Nascimento, an undergraduate student in Computer Science at the State University of Ceará (UECE), for developing the template that served as the foundation for the adaptation of this dissertation.

I would also like to thank all the professors who accompanied me from my undergraduate studies through my master's program. Beyond imparting knowledge, they played a crucial role in shaping my character and in the construction of my identity as a researcher.

Finally, I would like to acknowledge the Foundation for the Support of Technical Services, Education, and Research Promotion (FASTEF), in particular the Alan Turing Laboratory (ATLab), for providing an environment that fostered research and innovation. Through this laboratory, I had the opportunity to participate in research and development projects that significantly contributed to my academic and professional growth throughout the master's program.

"We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time."

(**T. S. Eliot**)

ABSTRACT

Understanding how knowledge about source code is distributed among contributors is critical to ensuring software quality and project sustainability over time. Established techniques, such as Truck Factor and authorship metrics, are based on contribution frequency and file ownership, but do not consider developers' behavioral patterns when navigating the code base. This work adapts the Returners and Explorers (R&E) framework, originally proposed in the field of human mobility, to characterize developers' interaction with source code based on orbital metrics. Using the concepts of code orbit radius (r_{co}) and its variant located in the k most modified files ($r_{co}^{(k)}$), developers can be classified as returners, explorers, or tourists, according to the dispersion of their modifications. The empirical analysis covers 20 open-source projects from the Apache Software Foundation, identifying that returners tend to concentrate on specific regions of the code, while explorers have a more comprehensive performance. Although often overlooked, tourists represent a significant portion of contributors, especially in peripheral files. A comparison between R&E profiles and expertise-based models, such as DOA, DOE, and Truck Factor, reveals that explorers often coincide with expert developers, while returners, although specialized, are sometimes underestimated. The R&E framework complements existing approaches by capturing behavioral dimensions of knowledge distribution, offering new perspectives on developer roles, collaboration patterns, and cognitive risks in large-scale software projects.

Keywords: software maintenance tools; collaboration in software development; software post-development issues; empirical software validation; software development techniques.

RESUMO

Compreender como o conhecimento sobre o código-fonte está distribuído entre os colaboradores é fundamental para garantir a qualidade do software e a sustentabilidade de projetos ao longo do tempo. Técnicas consolidadas, como o Truck Factor e métricas de autoria, baseiam-se em frequência de contribuições e propriedade de arquivos, mas não consideram os padrões comportamentais dos desenvolvedores ao navegar pela base de código. Este trabalho adapta o framework Returners and Explorers (R&E), originalmente proposto no campo da mobilidade humana, para caracterizar a interação de desenvolvedores com o código-fonte a partir de métricas orbitais. Por meio das noções de raio de órbita de código (r_{co}) e sua variante localizada nos k arquivos mais modificados ($r_{co}^{(k)}$), é possível classificar os desenvolvedores como returners, explorers ou turistas, de acordo com a dispersão de suas modificações. A análise empírica abrange 20 projetos open-source da Apache Software Foundation, identificando que returners tendem a se concentrar em regiões específicas do código, enquanto explorers apresentam atuação mais abrangente. Apesar de frequentemente negligenciados, os turistas representam uma parcela expressiva de colaboradores, especialmente em arquivos periféricos. A comparação entre os perfis R&E e modelos baseados em expertise, como DOA, DOE e Truck Factor, revela que explorers frequentemente coincidem com desenvolvedores especialistas, ao passo que returners, embora especializados, são por vezes subestimados. O framework R&E se mostra complementar às abordagens existentes por capturar dimensões comportamentais da distribuição de conhecimento, oferecendo novas perspectivas sobre papéis de desenvolvedores, padrões de colaboração e riscos cognitivos em projetos de software em larga escala.

Palavras-chave: ferramentas de manutenção de software; colaboração no desenvolvimento de software; problemas pós-desenvolvimento de software; validação empírica de software; técnicas de desenvolvimento de software.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS

i, j	Indices of modified files
L	Set of files modified by a developer
n_i	Number of modifications performed on file i
N	Total number of modifications performed by the developer
N_k	Total number of modifications across the top-k most edited files
$d(i, j)$	Geodesic distance between files i and j
f^*	Developer's most frequently modified file
r_{co}	Global code orbit radius
$r_{co}^{(k)}$	Code orbit radius computed over the top-k files
$p^{(k)}$	Ratio between $r_{co}^{(k)}$ and r_{co}, used for behavior classification
k	Number of top-most edited files considered in the localized metric
k^*	Equilibrium point where the number of returners equals or exceeds the number of explorers

CONTENTS

1 INTRODUCTION

This dissertation investigates how developers navigate large-scale software systems and how these behavioral patterns reflect and affect knowledge distribution in collaborative projects. To this end, we adapt the *Returners and Explorers* (R&E) framework, originally developed to model human mobility, to characterize developers based on their interaction with source code. This chapter introduces the general context of software maintenance, presents the motivation and challenges that underpin the study, defines the research problem, objectives, and questions, and outlines the structure of the dissertation.

1.1 Context and Motivation

Software maintenance and evolution depend on recurring codebase changes (MENS *et al.*, 2005), essential for adding new features, enhancing existing functionalities, and fixing bugs. As software projects scale, especially with geographically dispersed teams, understanding how source code knowledge is distributed among developers becomes increasingly important (FERREIRA *et al.*, 2019). Such insights can enhance task assignment efficiency (e.g., identifying suitable developers for code reviews or bug fixes) and mitigate risks associated with knowledge concentration in a limited group of contributors, which could jeopardize project sustainability due to developers' turnover.

Existing methods for evaluating code knowledge concentration, like Truck Factor, typically rely on authorship and ownership metrics such as Degree of Authorship (DOA) (FRITZ *et al.*, 2014) and Degree of Expertise (DOE) (CURY *et al.*, 2024). These approaches predominantly use repository-level indicators of developer contributions but lack detailed insight into developers' interaction patterns across the codebase over time.

The Returners and Explorers (R&E) dichotomy emerged from computational studies of human mobility patterns, fundamentally changing how researchers understand individual movement behaviors in physical space (PAPPALARDO *et al.*, 2015). This framework distinguishes between two distinct behavioral profiles: *returners*, who exhibit recurrent visitation patterns to familiar locations, and *explorers*, who continuously seek novel destinations and demonstrate broader spatial dispersion. Unlike traditional mobility models that focus on aggregate flows or distance distributions, the R&E approach characterizes individuals based on the relationship between their global movement patterns and their localized activity around

frequently visited locations, typically measured using the radius of gyration metric.

The conceptual power of the R&E dichotomy has transcended its original domain, inspiring applications across diverse fields including urban planning, epidemiology, and social network analysis (BARBOSA *et al.*, 2018; SCHLÄPFER *et al.*, 2021). In urban studies, the framework has informed transportation infrastructure design by identifying commuter versus exploration patterns. Epidemiological research has leveraged R&E classifications to model disease spread dynamics, recognizing that returners and explorers exhibit different infection risk profiles (WESOLOWSKI *et al.*, 2012). These cross-disciplinary applications demonstrate the framework's potential as a general behavioral lens for understanding how individuals navigate complex systems, a principle we now extend to the domain of software development on open-source projects.

In this research, we adapt the R&E dichotomy framework to the software engineering context to characterize how developers interact with the codebase, examining whether their activities are predominantly localized (returners) or distributed across a wider area (explorers). We operationalize the concepts of spatial dispersion using the novel metrics "radius of code orbit" and "k-radius of code orbit", derived from the hierarchical structure of the codebase. We evaluate the adapted version of the R&E framework using developers' historical commit data from 20 long-lived open-source projects within the Apache Software Foundation ecosystem.

1.2 Problem Statement, Goals, and Research Questions

Problem Statement

Traditional expertise metrics capture *what* developers modified, but not *how* they navigate through the software project codebase. As a result, they overlook behavioral dimensions of code engagement that may be critical to understanding knowledge flow, risk exposure, and collaboration dynamics. There is a need for models that go beyond ownership to represent the behavioral profiles of developers in large-scale codebases.

Goals and Research Questions

The main goal of this evaluation is two-fold. First, gather empirical evidence on the extent to which the R&E dichotomy fits the open-source software development domain. Second, understand if the R&E classification can be used as a proxy to characterize source code

knowledge distribution between developers. Therefore, we raise the following research questions to guide our empirical evaluation.

RQ1. *To what extent the R&E dichotomy can be used to characterize the code change patterns in open-source software development*?

This research question investigates how the adapted Returners and Explorers (R&E) framework can be used to characterize code change patterns in open-source software development. In this context, code change patterns refer to how developers navigate, revisit, and explore the codebase over time, including the concentration or dispersion of changes across files and directories. By applying the R&E dichotomy to a set of 20 long-lived open-source projects from the Apache ecosystem, we aim to analyze the proportion of returners and explorers and examine how their activities contribute to codebase coverage, particularly in critical areas of the system.

RQ2. *To what extent can the R&E framework be used as a proxy for the distribution of code knowledge in open-source software projects*?

This question seeks to analyze the potential of the R&E framework as a proxy tool for inferring tacit knowledge about the codebase. The underlying hypothesis is that the codebase experts are composed of both returners (localized experts) and explorers (broadened experts). We use two existing strategies to compute the code experts and analyze how the proportion of returners and explorers changes when the parametric threshold of these approaches varies.

1.3 Scope and Assumptions

This study focuses on long-lived open-source software projects with publicly available version control histories, hierarchical directory structures, and sustained developer activity. These characteristics are required to enable the observation of developer mobility across the codebase and the identification of behavioral roles using the Returners and Explorers (R&E) framework. As a result, the findings of this study are primarily applicable to collaborative, repository-based software development contexts and should not be directly generalized to proprietary systems or projects with limited or fragmented change histories.

The analysis assumes that version control data provides a reliable proxy for developers' interaction with the codebase and that the directory structure reasonably reflects the organization of the system's modules. It is further assumed that recurring code modifications indicate familiarity with specific areas of the code, while modifications in previously unvisited areas reflect exploratory behavior. Finally, the study assumes that developers with sufficient

activity over time can be meaningfully classified within the R&E dichotomy.

1.4 Main Contributions and Limitations

Our findings indicate that the Returners and Explorers dichotomy captures consistent and measurable differences in how developers interact with the codebase across projects. While returners are characterized by repeated interactions within a limited set of files or directories, our analysis shows that this behavior manifests as a stable form of localized knowledge specialization. In contrast, explorers exhibit sustained engagement across multiple and previously unvisited regions of the codebase, contributing to broader codebase coverage and knowledge integration beyond what would be expected from transient activity alone.

Main Contributions

The main contributions of this dissertation are:

1. The conceptual and operational adaptation of the R&E framework to the domain of software engineering, introducing two new mobility-inspired metrics: the code orbit radius (r_{co}) and the k-orbit radius ($r_{co}^{(k)}$).
2. Empirical evaluation of the adopted framework through a dataset derived from 20 open source projects from the Apache ecosystem, demonstrating its utility in capturing nuanced developer behaviors.
3. Insights into the dynamics of knowledge distribution within open-source software projects, highlighting the critical yet previously unacknowledged role of transient contributors (tourists).
4. A new behavioral lens that complements traditional knowledge models and provides actionable insights for project managers and research on software sustainability.

This refined perspective not only complements existing authorship and ownership-based models but also offers project managers and stakeholders a richer understanding of collaborative dynamics, facilitating more informed strategies for knowledge management and team coordination.

Limitations

The approach relies on the availability and consistency of version control data and assumes a meaningful organization of code files in directory structures. It may be less applicable to projects with flat or irregular hierarchies, limited commit metadata, or where contribution patterns are not file-based (e.g., visual or configuration-driven development).

1.5 Thesis Outline

This dissertation is organized into five main chapters, structured as follows:

1. **Chapter 1 (*Introduction*)** presents the context and motivation of the study, defines the problem statement, research goals, and questions, and outlines the main contributions and limitations.
2. **Chapter 2 (*Theoretical Background and Related Work*)** discusses foundational concepts such as software knowledge models, Truck Factor, and the original Returners & Explorers (R&E) framework from human mobility, as well as related studies on developer expertise and knowledge inference.
3. **Chapter 3 (*Methodology*)** details the research design, including the adaptation of the R&E framework to code navigation, project selection, dataset structure, data cleaning and normalization, and the computation of dispersion, classification, and file criticality metrics.
4. **Chapter 4 (*Results and Discussion*)** presents the empirical findings, structured around RQ1 (R&E and code change patterns) and RQ2 (R&E and code knowledge), and interprets them in light of existing models, exploring behavioral perspectives on code knowledge and implications for software engineering and beyond.
5. **Chapter 5 (*Conclusion and Future Work*)** summarizes the main findings, presents the final remarks, and outlines promising directions for future research.

2 THEORETICAL BACKGROUND AND RELATED WORK

This chapter presents the theoretical foundations and prior research that support the development of this study. We begin by reviewing key concepts from the field of human mobility, with a focus on the behavioral distinction between residents and tourists. This discussion provides the basis for introducing the Returners & Explorers (R&E) framework, a model that classifies individuals based on spatial dispersion patterns. We then describe the formalization of the R&E framework and its potential for modeling behavioral dynamics. Finally, we review related work in software engineering, including models for estimating developer expertise and assessing knowledge concentration through Truck Factor analysis. This discussion situates our approach within the broader research landscape and highlights the gap that our behavioral framework aims to address.

2.1 Background

2.1.1 Human Mobility Patterns: An Overview

Human mobility research has emerged as a transformative field that has fundamentally reshaped our understanding of human behavior and decision-making processes (BARBOSA *et al.*, 2018). The availability of large-scale digital traces has enabled researchers to quantify individual movement behaviors with unprecedented precision (GONZÁLEZ *et al.*, 2008; SONG *et al.*, 2010), revealing complex patterns that govern how people navigate through space and make location choices in their daily lives.

The study of mobility patterns has become increasingly important for understanding human behavior, particularly as urbanization concentrates the majority of the world's population in cities. In urban environments, mobility research typically distinguishes between two fundamental population groups: residents and tourists. Despite the fruitful research outcomes in human mobility studies, most findings are drawn upon urban residents, while the behavioral characteristics of other population groups, such as tourists, remain underexplored. Residents exhibit structured, routine-based movement patterns that reflect long-term spatial commitments and familiarity with the urban landscape. Regular commuting travel demands represent predictable flows between residential areas and business areas, forming cyclical patterns that are highly predictable and recurring. Tourists, in contrast, display more erratic and exploratory behaviors

characterized by broader spatial coverage and temporal flexibility. Tourists have travel patterns that are focused on exploration rather than routine, lacking the temporal consistency and spatial anchoring that characterizes resident mobility. Contemporary tourists' movements in cities are no longer bound by rigid schedules or fixed itineraries, and their temporal and spatial flexibility is highlighted by their mobility patterns. This distinction is crucial because residents provide the stable behavioral foundation necessary for understanding systematic mobility dynamics, while tourists introduce transient variability that, while interesting in its own right, can obscure the underlying structural patterns that govern urban systems (XU *et al.*, 2021).

Traditional mobility studies focused primarily on aggregate flows and distance distributions but failed to capture the heterogeneity in individual navigation strategies among residents. This limitation became particularly evident when attempting to model how people balance exploration of new locations with exploitation of familiar places; a fundamental trade-off that governs both daily routines and people's long-term spatial behavior (SIMINI *et al.*, 2012).

2.1.2 Formalization of the Returners & Explorers Framework

The Returners and Explorers (R&E) framework emerged as a response to this analytical gap, providing a principled method to classify resident individuals based on their spatial dispersion patterns (PAPPALARDO *et al.*, 2015). The framework operationalizes two distinct behavioral profiles through the relationship between global and localized mobility metrics. Specifically, it compares an individual's overall radius of gyration with their radius of gyration computed over the most frequently visited locations. *Returners* are characterized by high recurrence to a limited set of familiar locations, resulting in spatial dispersion that is largely explained by their top-k most visited places. *Explorers*, conversely, exhibit broader spatial coverage with significant activity beyond their core locations, indicating a propensity for seeking novel destinations over time (PAPPALARDO *et al.*, 2015).

Formally, given a user who has visited a sequence of N locations, each identified by a spatial coordinate $\vec{r}_i$, the **radius of gyration** is defined as:

$$r_g = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|\vec{r}_i - \vec{r}_{cm}\|^2} \tag{2.1}$$

where $\vec{r}_{cm}$ is the center of mass of the trajectory, computed as the average position of all visited locations:

$$\vec{r}_{cm} = \frac{1}{N}\sum_{i=1}^{N}\vec{r}_i \tag{2.2}$$

To assess the concentration of mobility around the most visited locations, the localized version of this metric is computed by restricting the summation to the top-k most frequently visited places. The resulting **localized radius of gyration** is denoted $r_g^{(k)}$:

$$r_g^{(k)} = \sqrt{\frac{1}{N_k}\sum_{i=1}^{k} n_i \left\|\vec{r}_i - \vec{r}_{cm}^{(k)}\right\|^2} \tag{2.3}$$

where n_i is the number of visits to location i, $N_k = \sum_{i=1}^{k} n_i$ is the total number of visits among the top-k locations, and $\vec{r}_{cm}^{(k)}$ is the weighted center of mass restricted to these k locations.

This formulation enables a comparative analysis of global and localized dispersion patterns. The ratio between $r_g^{(k)}$ and r_g reflects the behavioral tendency of an individual toward recurrence or exploration. A high ratio indicates that the individual's movement is well explained by a few core locations (returner), while a low ratio suggests more diffused spatial behavior (explorer).

This dichotomy has proven particularly valuable for understanding temporal dynamics in mobility, revealing that most individuals exhibit mixed behaviors that can shift between returning and exploring phases depending on contextual factors (ALESSANDRETTI *et al.*, 2018). The framework's strength lies in its ability to capture behavioral tendencies rather than rigid classifications, making it adaptable to domains where similar exploration-exploitation trade-offs occur.

The conceptual parallels between human mobility in urban environments and developer navigation through software codebases are compelling. Just as urban residents develop routine patterns of movement while occasionally exploring new areas, software developers exhibit similar behavioral tendencies when interacting with codebases over time. This analogy suggests that the R&E framework, originally designed to understand human spatial behavior, can be meaningfully adapted to characterize how developers navigate the structured, hierarchical landscape of software systems.

2.2 Related Work

The distribution of source code knowledge among developers is a central concern in Software Engineering, with implications for collaboration, risk mitigation, and long-term maintainability. This chapter surveys prior research along two complementary dimensions: (i) models that estimate developer expertise using version control data, and (ii) methods for assessing knowledge concentration through Truck Factor analysis. While both lines of work have yielded effective strategies for identifying key contributors, they typically emphasize contribution volume rather than behavioral dynamics. In contrast, our study introduces a mobility-inspired framework to analyze how developers traverse the codebase, offering a new lens on knowledge dispersion.

2.2.1 Developer Expertise and Knowledge Inference

Understanding which developers are knowledgeable about which parts of a codebase is a longstanding concern in software engineering. Various techniques have been proposed to infer source code expertise by mining version control data, particularly commit histories. These techniques typically rely on features such as commit frequency, the number of modified lines, authorship of lines (e.g., via `git blame`), or file creation/change events (FRITZ *et al.*, 2014; AVELINO *et al.*, 2016; YAMASHITA *et al.*, 2015; WANG *et al.*, 2020).

Early models often focused on simple metrics, such as the number of commits per file, to approximate developer expertise. More sophisticated approaches, like the Degree of Authorship (DOA) (FRITZ *et al.*, 2014), combine multiple signals such as first authorship, change frequency, and contributions by others to generate a normalized expertise score. Later work incorporated additional factors, such as modification recency (LUCAS *et al.*, 2020) or churn (YAMASHITA *et al.*, 2015), and used these features to support expert recommendation (MONTANDON *et al.*, 2019) or developer-task assignment (WANG *et al.*, 2020).

Recent studies also explored machine learning classifiers trained on mined repository features to improve the accuracy of expert identification (CURY *et al.*, 2024). These models are particularly useful for quantifying the distribution of code knowledge and identifying potential knowledge silos. In this work, we adopt a complementary perspective. Instead of predicting expertise for a particular file or task, we analyze the frequency and breadth of file changes to characterize developer activity. This aligns with prior work on expert identification but shifts

focus from predictive modeling to behavioral profiling. The adapted Returners and Explorers (R&E) framework builds on this view and complements authorship-based models by emphasizing how developers traverse the codebase over time.

2.2.2 Truck Factor and Knowledge Concentration

A critical challenge in collaborative software development is preserving knowledge continuity as contributors leave a project. The agile community introduced the Truck Factor (TF), or Bus Factor, to quantify the minimal number of developers whose loss would critically impair a project (AVELINO *et al.*, 2016). A low TF indicates high knowledge concentration and greater risk, while a high TF suggests distributed expertise, often fostered by practices like collective code ownership. TF has gained traction in both open-source and industrial contexts as a proxy for assessing project resilience to team turnover.

Several algorithms have been proposed to estimate TF from version control data. A widely used approach by Avelino *et al.* (2016) employs the DOA metric (FRITZ *et al.*, 2014) to estimate ownership. A greedy algorithm then removes top authors until less than 50% of the files remain covered. This method, validated across 133 projects hosted on GitHub, aligned with developer perceptions in 84% of surveyed cases. Comparative studies, such as Ferreira *et al.* (2019), found it more accurate than alternatives based on blame or commit heuristics.

Building on this, Cury *et al.* (2024) proposed models that leverage repository-level metrics to improve expert identification. Notably, the Degree of Expertise (DOE) model is a linear alternative that integrates first authorship, recency of change, file size, and lines added. When used in TF estimation, DOE improves recall and captures impactful recent contributors often missed by DOA. These refinements enhance TF's utility as a proxy for knowledge distribution, offering a more current and nuanced view of developer expertise. However, TF estimation inherently depends on threshold parameters used to classify developers as file experts, regardless of whether expertise is derived from DOA or DOE. Prior studies have shown that reasonable threshold values can be defined for both open-source and industrial systems (AVELINO *et al.*, 2016; CURY *et al.*, 2024). Nevertheless, as with any threshold-based expertise model, the choice of these parameters may influence the resulting TF values and should be made with methodological care, taking into account the project characteristics.

2.3 Summary

This chapter reviewed the theoretical and empirical foundations that underpin this study. First, we examined human mobility research, highlighting the behavioral dichotomy between residents and tourists and its formalization through the Returners & Explorers (R&E) framework. The discussion emphasized how the balance between exploration and recurrence provides a powerful lens to model spatial dispersion and behavioral dynamics.

We then surveyed related work in software engineering, focusing on two complementary perspectives: developer expertise and knowledge inference models, which capture contribution intensity but often overlook navigational patterns, and Truck Factor analysis, which quantifies knowledge concentration but remains limited to static coverage of source code. Together, these approaches demonstrate the importance of understanding knowledge distribution but leave unaddressed the behavioral dynamics of how developers traverse and revisit the codebase.

By bridging these areas, this study positions the R&E framework as a novel perspective to study software development. The framework complements existing expertise and Truck Factor models by introducing metrics of spatial dispersion in code navigation. This connection establishes the theoretical foundation for the methodological adaptations described in the next chapter, where we operationalize the R&E framework to analyze developer behavior in real-world software projects.

3 METHODOLOGY

This chapter presents the methodological approach employed in this research to adapt, operationalize, and evaluate the Returners & Explorers (R&E) framework in the context of software engineering. Our strategy is divided into three major components: (i) the adaptation of the R&E framework to model developer behavior in codebases; (ii) the construction and preprocessing of an empirical dataset from real-world open-source projects; and (iii) a set of analytical procedures designed to answer our research questions through the quantification of developer dispersion and comparison with traditional expertise models.

3.1 From Human Mobility to Code Navigation: Adaptation of the Returners & Explorers Framework

The adaptation of the Returners and Explorers (R&E) framework to software engineering seeks to analyze developer movement patterns across the codebase by drawing analogies with human mobility. In the original formulation by Pappalardo *et al.* (2015), individuals are classified according to the spatial dispersion of their visits: *returners* frequently return to the same places, while *explorers* tend to move through a broader and more diverse set of locations.

When transposed to the software domain, these concepts are mapped as follows: code files represent the "locations", a "visit" corresponds to a developer modifying a file through a commit, and developers are individuals who can play the role of returner or explorer. This analogy allows us to frame developer contributions not just as discrete events, but as traces of behavioral navigation through the project's codebase structure. Understanding whether developers concentrate their changes locally or distribute them widely can reveal insights into knowledge specialization, code ownership, and collaborative practices in a software project repository.

3.1.1 Geodesic Distance: A Structural Proxy for Spatial Dispersion

In the software domain, spatial coordinates are not meaningful. Instead, we define proximity using the *geodesic distance* $d(i, j)$ between two files i and j, computed as the number of upward and downward transitions in the directory tree required to traverse from one to the other.

To enable these computations, each file in the repository is modeled as a node in a

rooted directory tree. The structural distance between files, called geodesic distance, is computed as the number of steps needed to reach from one file to another by moving up or down the hierarchy. This method is not only interpretable and efficient but also agnostic to the specific structure of the system, making it applicable across heterogeneous codebases. Algorithm 1 presents this procedure.

Algorithm 1: Geodesic distance computation between two file paths

Input: Paths $\texttt{Path}_1$, $\texttt{Path}_2$

Output: Geodesic distance between the two paths

begin

  Split $\texttt{Path}_1$ and $\texttt{Path}_2$ by directory separator;

  Initialize common prefix counter $common \leftarrow 0$;

  for $i \leftarrow 0$ *to the minimum length of both paths* **do**

    if $parts_1[i] = parts_2[i]$ **then**

      $common \leftarrow common + 1$;

    end

    else

      break the loop;

    end

  end

  $up\_steps \leftarrow$ length of $parts_1 - common$;

  $down\_steps \leftarrow$ length of $parts_2 - common$;

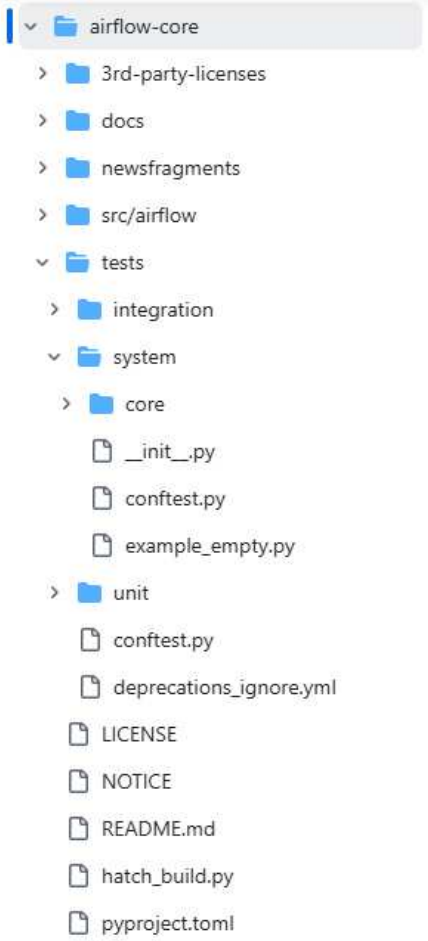
  return $up\_steps + down\_steps$;

end

To illustrate the procedure, consider the two file paths *airflow-core/hatch_build.py* and *airflow-core/tests/system/conftest.py*. Applying Algorithm 1, both paths are first split by the directory separator, resulting in $\text{Path}_1 \rightarrow$ [“airflow-core”, “hatch_build.py”] and $\text{Path}_2 \rightarrow$ [“airflow-core”, “tests”, “system”, “conftest.py”].

The longest common prefix consists of one element (“`airflow-core`”), which means both files share the same top-level directory. From this common path, the first file `hatch_build.py` is located one level deeper, so we need a single upward step to return to the common directory ($up\_steps = 1$). Conversely, to reach the second file `conftest.py`, we must traverse from the common path through two intermediate directories (`tests` and `system`) before

Figure 1 – Example of a directory tree from the `airflow-core` repository.



Source: https://github.com/apache/airflow/tree/main/airflow-core.

arriving at the file itself. This results in three downward steps ($down\_steps = 3$). The geodesic distance is therefore the sum of both components:

$$d(\texttt{airflow-core/hatch\_build.py}, \texttt{airflow-core/tests/system/conftest.py}) = 1+3 = 4.$$

This example illustrates how the geodesic distance reflects structural proximity in a directory tree: although both files belong to the same top-level module (`airflow-core`), their relative separation in the hierarchy yields a distance of four steps.

3.1.2 Radius of Code Orbit: Adaptation of Radius of Gyration to Codebases

To operationalize this analogy, we map software artifacts and developer actions to the spatial constructs of the R&E model. Each source code file is treated as a symbolic "location," and every file modification, such as a commit, is considered a "visit." Developers are thus conceptualized as agents whose navigational trajectories unfold over the hierarchical structure of the codebase. This perspective allows us to interpret code contributions not as isolated events,

but as patterns of behavioral movement across modular components of the system. Such framing provides insights into specialization, ownership, and the breadth of developer engagement.

To quantify this behavior, we introduce the **radius of code orbit** (r_{co}), a metric adapted from the original *radius of gyration* (r_g) (2.3), which in physical and mobility sciences measures the spatial dispersion of visited locations.

The developer's **global radius of code orbit** is then defined as:

$$r_{co} = \sqrt{\frac{1}{N} \sum_{i \in L} n_i \cdot d(i, f^*)^2} \tag{3.1}$$

where L is the set of files modified by the developer, n_i is the number of modifications to file i, and f^* is the developer's most frequently modified file, serving as the behavioral anchor (analogous to a home location).

To assess the concentration of activity, we define the localized dispersion metric, the **k-radius of code orbit**:

$$r_{co}^{(k)} = \sqrt{\frac{1}{N_k} \sum_{i=1}^{k} n_i \cdot d(i, f^*)^2} \tag{3.2}$$

where N_k is the total number of modifications to the top-k most edited files.

The behavioral classification is derived from the ratio:

$$p^{(k)} = \frac{r_{co}^{(k)}}{r_{co}}$$

and interpreted as:

$$\text{Developer} = \begin{cases} k\text{-returner}, & \text{if } p^{(k)} > \frac{1}{2} \\ k\text{-explorer}, & \text{otherwise} \end{cases}$$

This rule reflects the degree to which a developer's dispersion is explained by a small core of files. A k-returner exhibits focused behavior, while a k-explorer navigates broadly across the system.

3.1.3 Equilibrium Point and Classification Stability

As the value of k increases, the top-k files begin to account for a larger share of a developer's total code modifications. Consequently, the localized dispersion $r_{co}^{(k)}$ approaches the global dispersion r_{co}, and the ratio $p^{(k)} = \frac{r_{co}^{(k)}}{r_{co}}$ tends to increase. This implies that even developers with dispersed behaviors can eventually be classified as returners for sufficiently large values of k.

To achieve a balanced representation of both behavioral extremes, those who consistently revisit specific modules and those who broadly explore the codebase, we adopt the notion of an **equilibrium point** k^*, originally proposed by PAPPALARDO *et al.* (PAPPALARDO *et al.*, 2015). This value corresponds to the integer k that minimizes the difference between the number of returners and explorers:

$$k^* = \arg\min_{k \in \mathbb{N}} \left| \#\text{returners}^{(k)} - \#\text{explorers}^{(k)} \right|$$

By definition, k^* is not necessarily the point where the number of returners exceeds that of explorers. Instead, it identifies the configuration in which the two groups are most evenly represented. Intuitively, this point captures the scale at which behavioral differences are most discernible, avoiding both over-fragmentation caused by very small values of k and over-aggregation induced by substantial values.

Rather than arbitrarily fixing k, the equilibrium point allows the classification to emerge from the data itself, promoting a stable and unbiased characterization in which neither exploratory nor specialized behaviors are systematically favored.

Table 1 summarizes the main variables and mappings used in the adaptation of the R&E framework to the software domain.

Table 1 – Mapping of the R&E framework to the software domain.

Variable	Description
i	Full name of the i-th modified file
L	Set of files modified by a developer
n_i	Number of edits made to the i-th file
N	Total number of modifications by the developer
$d(i, j)$	Geodesic distance between files i and j based on the directory tree
f^*	Developer's most frequently modified file (reference point)

Source: The Author.

3.2 Dataset Construction and Description

This section details the construction, structure, and refinement of the dataset used to evaluate the adapted Returners & Explorers (R&E) framework. Our goal was to assemble a large-scale, diverse, and representative dataset that captures developer activity and code evolution in mature, real-world projects. We describe the rationale and process for project selection, outline the organization of the collected data, and present the cleaning and normalization steps applied to ensure analytical reliability and consistency.

3.2.1 Project Selection

We choose projects from the Apache Software Foundation (ASF) to serve as subjects in our empirical investigation. We have decided to use projects from the ASF because they are open-source, mature, well-documented, have a clear governance model, rely on a big community collaboration, and are widely used to build new systems and provide support for business operations, indicating a certain degree of reliability from both developers' and companies' perspectives. All this makes these projects a good fit for our study.

We adopt two sets of criteria to select those projects. The first set is based on public access, longevity, activity, and popularity. The second set examines the domain of each project. For convenience, we have chosen five domain categories officially classified by the ASF as big-data, database, cloud, and library. Using different categories, we can evaluate whether our results are consistent across project domains. Thus, from the projects that matched the first set of criteria in the ASF's GitHub organization, we selected the five most popular of each domain, comprising a total of twenty projects.

We use the GitHub search query "`org:apache archived:false visibility:public pushed:>2025-03-31 created:<2019-12-31 stars:>500 sort:stars`" to automate the first set of selection criteria. This query searches for non-archived, public, and active (with pushed changes in the last month) repositories with more than 500 stars that are at least five years old. This query was performed on April 1, 2025, and retrieved about 150 repositories. Therefore, we applied the second set of criteria to select the 20 target repositories.

To ensure diversity in software domains, we applied a second filter based on ASF's project classification. We selected the five most popular repositories from each of four major categories: *big-data*, *database*, *cloud*, and *library*, resulting in 20 projects total. Table 2 summa-

rizes the selected projects, including domain, project name, creation year, GitHub popularity (stars), number of commits, and number of unique human contributors (after bot filtering and identity unification).

Table 2 – Selected Projects from the Apache Software Foundation.

Domain	Project	Started At	#Stars ($\times 10^3$)	#Commits	#Contributors
big-data	Airflow	2014	44	29,551	2,481
	Hadoop	2009	15	27,752	671
	Flink	2012	24	36,728	1,293
	Spark	2010	41	44,127	2,479
	Kafka	2012	30	15,501	1,242
database	Arrow	2016	15	17,416	1,041
	Cassandra	2009	9	30,770	455
	HBase	2007	5	20,517	570
	Hive	2008	5	17,624	501
	IoTDB	2017	5	12,303	254
cloud	Camel	2007	5	74,839	1,065
	CloudStack	2010	2	37,586	493
	CouchDB	2008	6	13,955	148
	Ignite	2014	4	29,312	370
	Libcloud	2009	2	10,219	426
library	bRPC	2017	17	3,356	197
	ECharts	2013	63	9,536	263
	Hudi	2016	5	6,279	474
	Lucene	2001	3	38,045	440
	Thrift	2008	10	7,137	519

Source: The author based on data from the Apache Software Foundation.

The dataset was assembled through an automated mining pipeline implemented in Python. For each selected project, the complete Git history was cloned and parsed using the PyDriller framework (SPADINI *et al.*, 2018). Two primary data artifacts were generated per project: The commit-level file contains high-level metadata for each commit, including hash, author and committer information (names and emails), timestamps, and code delta metrics such as size, complexity, and interface changes. The file-level file records detailed information on each file affected by a commit, including the file path before and after the change, type of change (e.g., ADD, DELETE, MODIFY, RENAME), number of lines added or removed, and source code metrics such as lines of code (LOC), cyclomatic complexity, and token counts.

3.2.2 Data Cleaning and Normalization

To ensure high-quality analysis, we applied a series of preprocessing steps designed to clean, normalize, and standardize the dataset across projects:

1. **Author Identity Unification:** Developers were initially identified using `(name, email)` pairs. We unified aliases by grouping entries with similar emails and selecting the most frequent or longest name as canonical. Additional heuristics were used to resolve punctuation variations in usernames.
2. **Anonymization:** Each unique developer identity was replaced with a synthetic, anonymized identifier to preserve privacy.
3. **Source Code Filtering:** Only changes involving source code files were retained (`IsSrcFile = 1`), excluding commits affecting documentation, tests, or configuration scripts that might distort developer dispersion metrics.
4. **Rename Resolution:** To track logical file identity over time, rename chains were resolved recursively, allowing renamed files to be treated as a continuous entity in dispersion calculations.
5. **Delete Handling:** Deleted files were not discarded; instead, their `FileOldPath` was retained, ensuring that all modification traces were preserved for spatial dispersion analysis.
6. **Bot Removal:** Contributions from automated agents (e.g., CI bots, formatters) were filtered using a curated list of known bot accounts.

Table 3 details the file extensions considered during the filtering step. Only files with extensions associated with source code were retained, while documentation-related extensions were excluded from the analysis.

Table 3 – File extensions considered for source code and documentation filtering.

Source Code Extensions	**Documentation Extensions**
.clj, .scala, .java, .py, .sc, .js, .jsx, .ts, .tsx	.md, .txt, .adoc, .asciidoc, .rst
.c, .cpp, .cc, .cxx, .h, .hpp, .cs, .rb, .go, .groovy	.html, .htm, .xml
.pl, .pm, .t, .pod, .sh, .php, .swift, .kt, .kts	
.rs, .r, .lua, .asm, .s, .ex, .exs, .erl, .hrl	
.fs, .fsi, .fsx, .hs, .lhs, .nim, .zig, .bat, .cmd	
.ps1, .psm1, .mk, .gradle, .bazel, .bzl, .dart, .m, .mm	

Source: The author.

These steps resulted in a curated dataset of human developer-file interactions that more accurately reflects intentional navigation behavior over the codebase.

3.3 Computation of Code Dispersion Metrics

This section describes how the adapted Returners & Explorers (R&E) framework was operationalized to quantify developer navigation behavior within software projects. Specifically, we computed structural dispersion metrics and behavioral classifications for all contributors, enabling project-level and file-level comparative analyses. These metrics form the foundation for answering our research questions.

3.3.1 Behavioral Classification of Developers

Building on the R&E adaptation described in Section 3.1, we applied the behavioral classification scheme to *resident developers*, those who modified at least one file more than once. Developers who performed only one change per file were labeled as *tourists* and excluded from the dispersion-based analysis, though they were retained for baseline comparisons.

For each resident, we reused the metrics r_{co}, $r_{co}^{(k)}$, and $p^{(k)}$ (as defined previously) to assign returner or explorer labels. We also determined the **equilibrium point** k^* for each project, representing the minimal k where returners outnumber or equal explorers. This threshold was used for project-level behavioral profiling.

3.3.2 Project-Level and Developer-Level Metrics

To contextualize behavioral classifications and support cross-project comparisons, we computed several descriptive metrics for each project: **#Authors**, the number of unique developers who modified at least one source file; **#Files**, the number of distinct source code files; **Aut/File**, the ratio between the number of developers and the number of source files; **#R&E**, the total number of returners and explorers (i.e., residents); **%R&E**, the proportion of authors classified as residents; **#T** and **%T**, the absolute and relative number of tourists; **#R** and **#E**, the number of returners and explorers, respectively; and **%R** and **%E**, the proportion of residents who are returners and explorers.

These metrics provide an overview of each project's contributor dynamics and behavioral diversity, allowing for detailed comparisons across different codebases.

3.3.3 Knowledge Proxies and Expert Role Distribution

To assess the relationship between R&E profiles and code expertise, we employed two well-established models of developer knowledge: **Degree of Authorship (DOA)** (FRITZ *et al.*, 2014) and **Degree of Expertise (DOE)** (CURY *et al.*, 2024). For each developer, both DOA and DOE metrics were computed at the file level. We then analyzed the distribution of experts across behavioral categories, considering **#DOA-Experts** and **#DOE-Experts**, the number of developers recognized as file-level experts by each model; **#DOA-Experts-R**, **-E**, and **-T**, the distribution of DOA experts among returners, explorers, and tourists; and **#DOE-Experts-R**, **-E**, and **-T**, the distribution of DOE experts across R&E profiles. These metrics provide a comparative basis to evaluate whether returners or explorers hold more prominent expert roles within the projects.

3.3.4 Truck Factor Composition and Critical Knowledge Holders

To investigate how behavioral profiles align with critical knowledge roles, we computed the **Truck Factor (TF)** of each project using both DOA and DOE metrics. The TF was derived using the greedy algorithm proposed by AVELINO *et al.* (AVELINO *et al.*, 2016), which iteratively removes top-ranked experts until less than half of the files remain covered. We report **DOA-TF** and **DOE-TF** as the Truck Factor size using DOA and DOE, respectively; **DOA-TF-R**, **-E**, and **-T** as the composition of the DOA-based TF in terms of R&E profiles; and **DOE-TF-R**, **-E**, and **-T** as the composition of the DOE-based TF in terms of R&E profiles. This enables us to assess whether critical knowledge holders in a project tend to exhibit returner or explorer behavior.

3.4 File Criticality Metrics

At the file level, we calculated metrics to capture structural and historical importance. These include **#Changes**, representing the number of commits that modified the file; **AMLOC**, the accumulated number of modified lines of code over the file's history; and **AvgCyC**, the average McCabe's cyclomatic complexity across file revisions. These metrics have been used to identify high-risk or high-importance files within each project (SOUSA *et al.*, 2025) and to evaluate how well they are covered by developers of each behavioral type. All calculations were applied consistently across the 20 projects to ensure comparability and reproducibility in our

analysis.

3.5 Restricted Classification: Filtering Borderline Cases

While the equilibrium-based classification enables a balanced separation of returners and explorers, it does not account for the strength or confidence of these behavioral profiles. Developers situated near the decision boundary, i.e., those with $p^{(k)}$ values close to 0.5, may exhibit transient or mixed behaviors, making their classification more sensitive to noise or small changes in activity.

To improve analytical robustness and isolate the most characteristic representatives of each behavioral category, we define a **restricted classification**. This approach applies stricter thresholds to the dispersion ratio $p^{(k)} = \frac{r_{co}^{(k)}}{r_{co}}$, aiming to exclude borderline developers and retain only those whose navigation behavior aligns with either returner or explorer patterns.

The classification is formalized as:

restricted returners: $p^{(k)} > 0.75$ restricted explorers: $p^{(k)} < 0.25$

Importantly, tourists are not directly affected by this filtering, as they are defined separately as developers whose file modification pattern does not meet the minimum criteria for dispersion-based classification.

3.6 Summary

This chapter detailed the methodological approach used to adapt and operationalize the Returners & Explorers (R&E) framework for the software engineering domain. We first described the conceptual adaptation from human mobility to code navigation, introducing the geodesic distance as a structural proxy for spatial dispersion and defining the radius of code orbit as the central metric for characterizing developer behavior. The classification scheme, based on the ratio between global and localized dispersion, enabled us to distinguish returners and explorers, while the equilibrium point ensured stability in these classifications.

We then presented the construction of the empirical dataset, which combined 20 diverse and mature Apache Software Foundation projects. The dataset was carefully curated through author identity unification, anonymization, source code filtering, rename resolution,

delete handling, and bot removal, resulting in a consistent and high-quality corpus of developer-file interactions.

Finally, we outlined the set of analytical procedures applied to this dataset. These included the computation of behavioral classifications, project-level and developer-level metrics, integration with knowledge proxies (DOA and DOE), assessment of Truck Factor composition, and evaluation of file-level criticality. Together, these methodological steps provide a comprehensive foundation for answering our research questions.

4 RESULTS AND DISCUSSION

This chapter presents the results of applying the Returners & Explorers (R&E) framework to the analyzed projects. The analysis is structured in two complementary parts. The first, referred to as the global classification, considers all developers classified according to their project-specific equilibrium point k. The second, the restricted classification, focuses only on the most representative cases of each behavioral profile, applying stricter thresholds to filter borderline instances.

Both classifications are examined under two different scopes. The first considers the entire codebase of each project, providing a comprehensive view of engagement across all components. The second focuses on the 10 most critical files, selected according to a composite metric that combines change frequency, accumulated lines of code (AMLOC), and average cyclomatic complexity (AvgCyC). Combining the two classification approaches with these analysis scopes makes it possible to contrast overall participation patterns with those observed in the project's most strategically important files.

4.1 R&E and Code Change Patterns (RQ1)

To address RQ1, we classified developers into three behavioral profiles: returners, explorers, and tourists. Returners and explorers were identified based on the project-specific value of k, which represents the equilibrium point where the population of returners and explorers is approximately balanced. Tourists, on the other hand, are developers whose activity was insufficient for meaningful classification (e.g., too few commits or too low coverage).

Table 4 summarizes key metrics for the 20 analyzed Apache projects, including the total number of developers, number of source files, the authors-per-file ratio (`Aut/File`), the number and percentage of developers classified as returners and explorers (`#R&E`, `%R&E`), the number and percentage of tourists (`#T`, `%T`), and the equilibrium point k adopted per project. Additionally, we report the distribution between returners and explorers within the classified group.

As the value of k increases, the classification dynamics follow a consistent pattern across all analyzed projects. The number of returners grows monotonically, while the number of explorers declines, reflecting the progressive concentration of developer activity as the analysis progressively incorporates more central files.

Table 4 – Summary of Project Metrics.

Project	#Authors	#Files	Aut/File	#R&E	#T	%R&E	%T	k	R&E			
									#R	%R	#E	%E
Airflow	2478	6341	0.39	592	1886	23.9	76.1	3	321	54.2	271	45.8
Arrow	1036	4223	0.25	331	705	31.9	68.1	4	162	48.9	169	51.1
bRPC	195	1138	0.17	70	125	35.9	64.1	3	43	61.4	27	38.6
Camel	1064	23035	0.05	446	618	41.9	58.1	4	220	49.3	226	50.7
Cassandra	454	5333	0.09	146	308	32.2	67.8	7	73	50.0	73	50.0
CloudStack	487	8074	0.06	262	225	53.8	46.2	5	137	52.3	125	47.7
CouchDB	147	979	0.15	68	79	46.3	53.7	3	33	48.5	35	51.5
ECharts	260	830	0.31	84	176	32.3	67.7	3	43	51.2	41	48.8
Flink	1284	15390	0.08	428	856	33.3	66.7	7	226	52.8	202	47.2
Hadoop	666	13746	0.05	295	371	44.3	55.7	7	141	47.8	154	52.2
HBase	566	5420	0.10	233	333	41.2	58.8	5	111	47.6	122	52.4
Hive	499	9580	0.05	231	268	46.3	53.7	6	124	53.7	107	46.3
Hudi	473	3790	0.12	163	310	34.5	65.5	6	79	48.5	84	51.5
Ignite	369	12499	0.03	192	177	52.0	48.0	10	97	50.5	95	49.5
IoTDB	253	6714	0.04	162	91	64.0	36.0	8	85	52.5	77	47.5
Kafka	1236	6049	0.20	341	895	27.6	72.4	5	161	47.2	180	52.8
Libcloud	424	708	0.60	223	201	52.6	47.4	2	132	59.2	91	40.8
Lucene	439	5912	0.07	172	267	39.2	60.8	7	92	53.5	80	46.5
Spark	2478	8035	0.31	819	1659	33.0	67.0	4	459	56.0	360	44.0
Thrift	517	1682	0.31	146	371	28.2	71.8	3	68	46.6	78	53.4

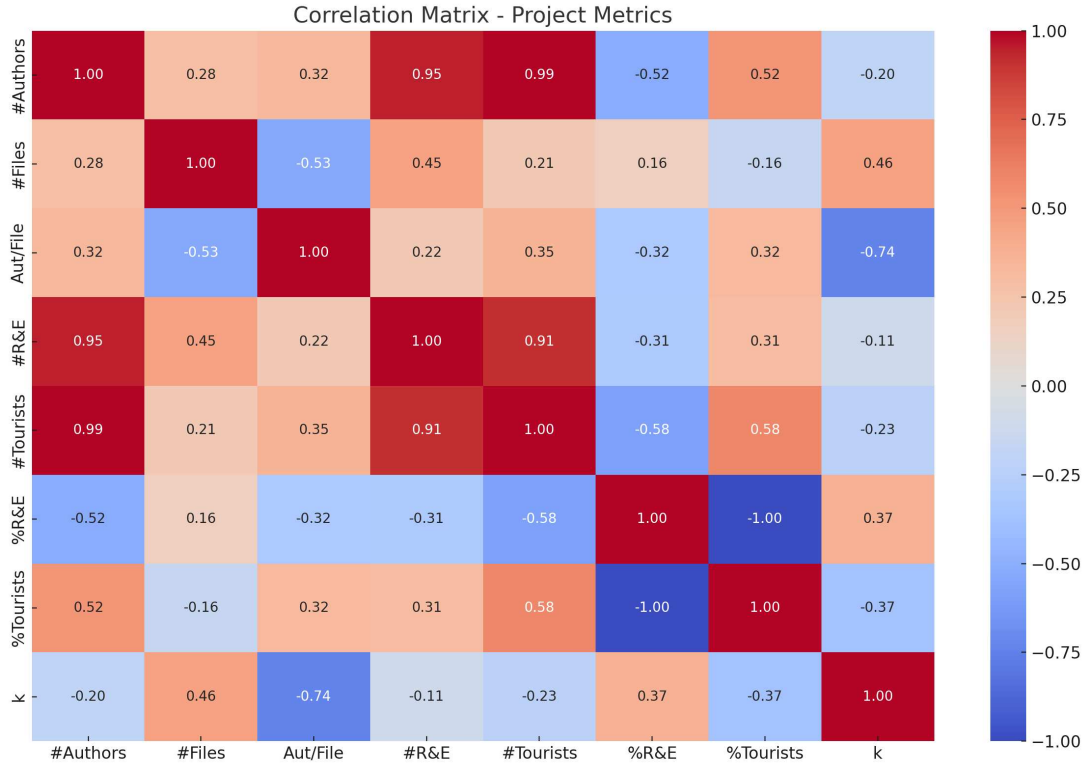
R&E: Returners & Explorers; **T**: Tourists; **k**: # top-k files used in classification; **R**: Returners; **E**: Explorers.
Source: The author.

Although the absolute growth rates vary across projects, the qualitative behavior remains remarkably stable. This consistency reinforces the rationale for adopting a project-specific equilibrium point k^*, which captures the scale at which localized and dispersed behaviors are most balanced. For completeness, the per-project evolution curves are provided in Appendix A.

To better understand the dynamics behind the equilibrium point k, we conducted a correlation analysis between the key metrics extracted from each project. As shown in Figure 2, the most notable pattern is a strong negative correlation between the equilibrium value k and the authors-per-file ratio. This indicates that in projects where more developers tend to work on the same files, suggesting higher overlap and shared ownership, the behavioral balance between returners and explorers is achieved at lower values of k. Conversely, in projects with a more distributed or siloed contribution pattern, the convergence of roles requires more iterations of file centrality filtering, resulting in higher k values.

Most projects exhibit a high prevalence of tourists—ranging from 36.0% in IoTDB to 76.1% in Airflow—suggesting either high contributor churn or a sizable group of peripheral

Figure 2 – Correlation matrix between project-level metrics and the equilibrium point k.



Source: The author.

contributors. In most cases, returners and explorers represent a minority of the contributor base, with notable exceptions such as CloudStack, Ignite, IoTDB, and Libcloud, where the classified group comprises the majority.

While the equilibrium point k^* provides a principled way to balance returners and explorers at the project level, it does not distinguish between strongly characteristic behaviors and borderline cases. In practice, many developers exhibit mixed or transient activity patterns that place them near the classification threshold. To better understand how robust the observed behavioral patterns are, we contrast the global classification with a restricted view that retains only the most representative returners and explorers.

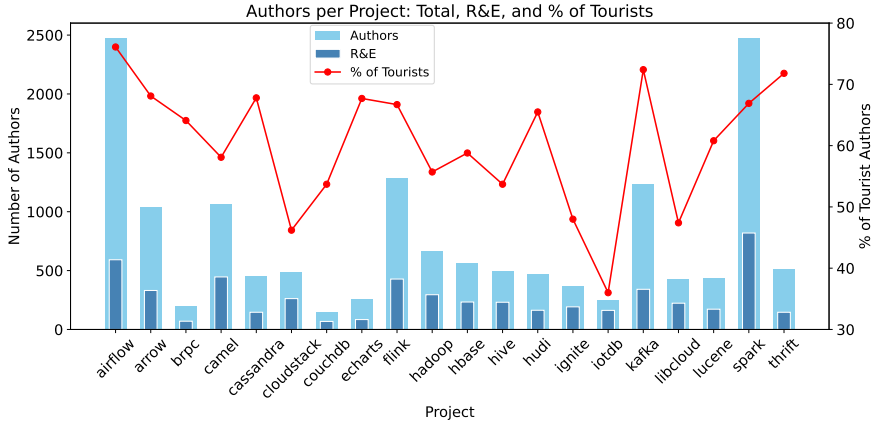
In addition, behavioral roles may manifest differently depending on the strategic importance of the code being modified. Changes to peripheral files may reflect onboarding or sporadic activity, whereas contributions to critical files are more likely to indicate sustained engagement and accumulated knowledge. For this reason, we analyze both classifications under two complementary scopes: the entire codebase and the top-10 most critical files. This comparison allows us to assess how developer roles shift when focusing on the most central components of the system.

Figure 3 highlights two consistent patterns across projects. First, the restricted

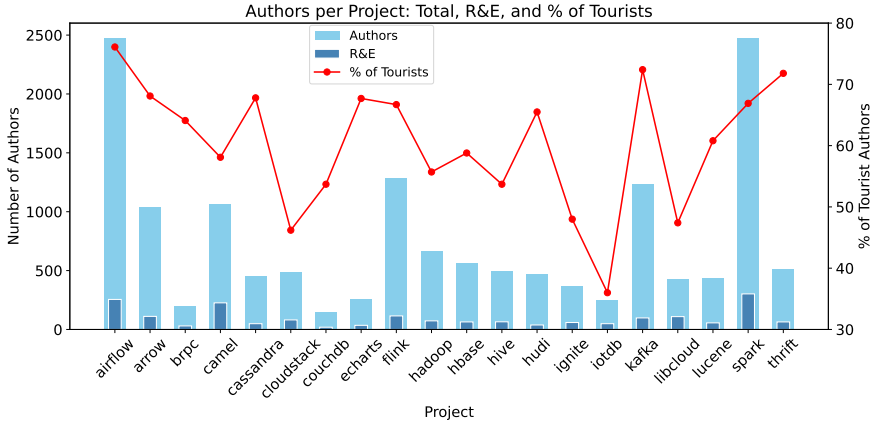
Figure 3 – Comparison of Global and Restricted classifications of Authors, Returners & Explorers (R&E), and the proportion of tourists across projects. The top panel shows results for the entire codebase, while the bottom panel focuses on the top-10 most critical files.

Entire Codebase

Global Classification

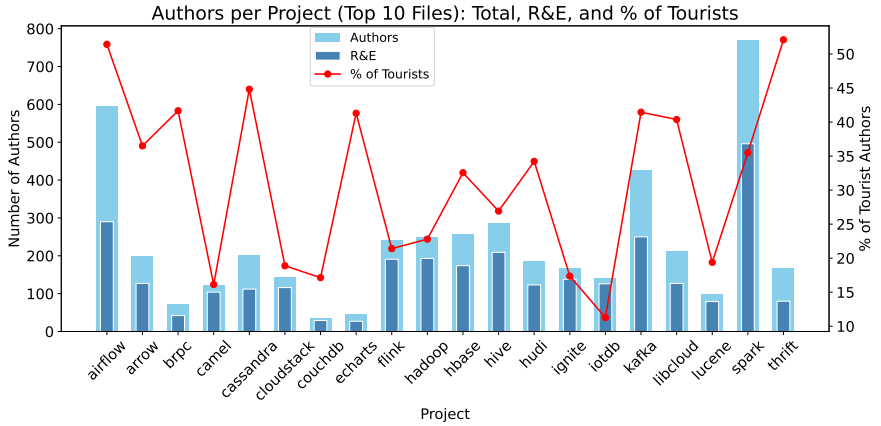


Restricted Classification

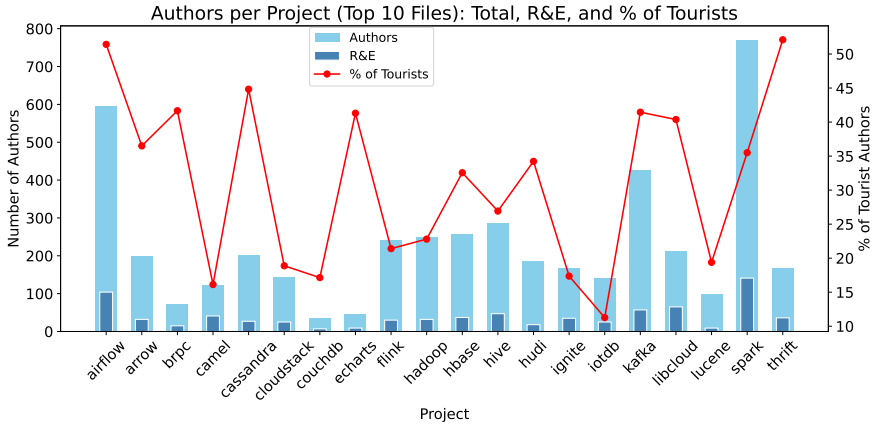


Top-10 Most Critical Files

Global Classification



Restricted Classification



Source: The author.

classification substantially reduces the number of developers classified as returners or explorers. This reduction is expected, as stricter thresholds remove borderline cases; however, its magnitude varies considerably across projects. Systems with more stable contribution patterns retain a sizable core of classified developers, whereas projects with higher churn experience a sharper contraction, indicating that a large fraction of contributors operate near the behavioral boundary.

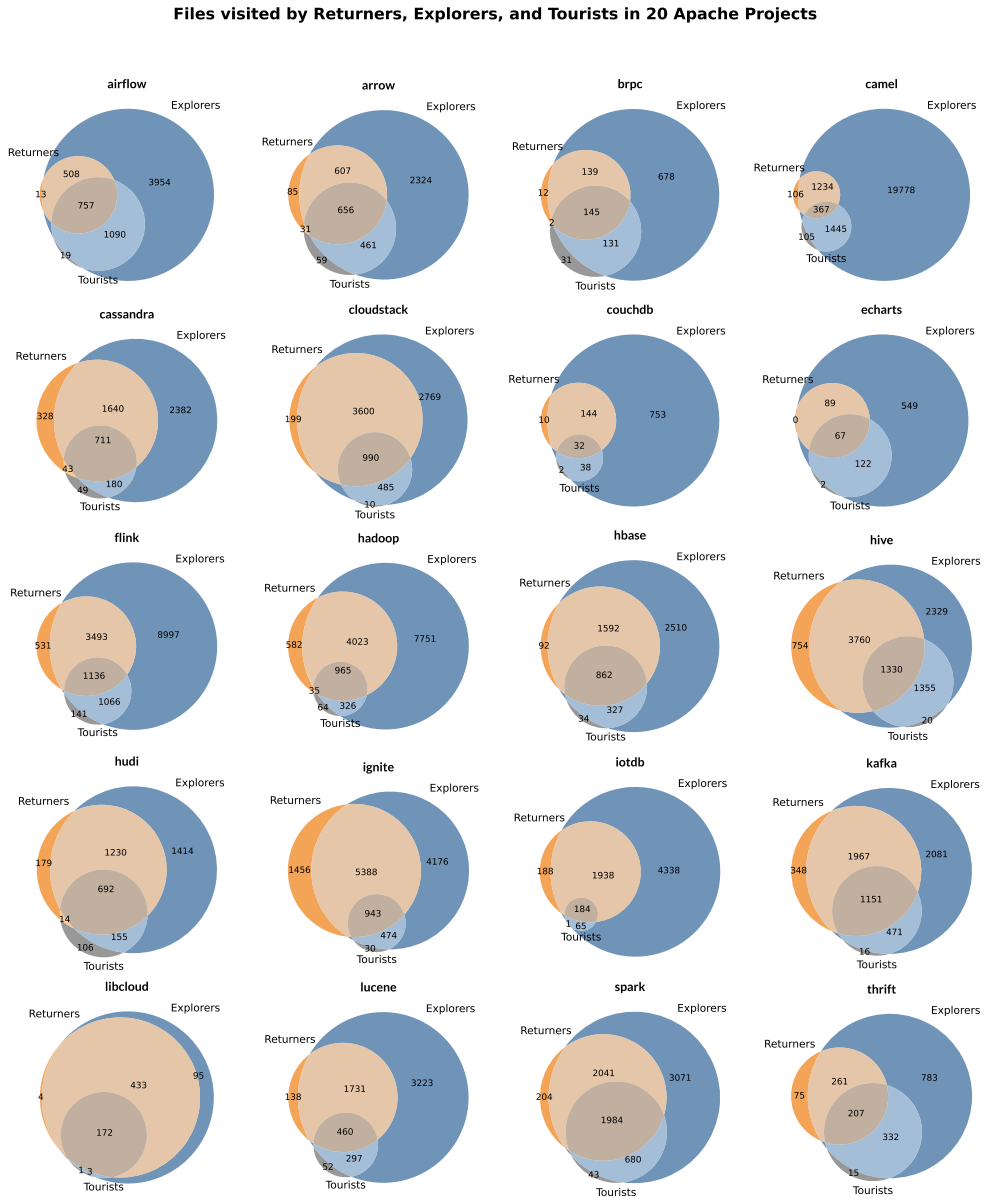
Second, the focus on the top-10 most critical files reveals a pronounced shift in role composition. Under the global classification, tourists already become less prevalent in critical files, suggesting that core components attract more persistent contributors. This effect is further amplified under the restricted classification, where returners and explorers dominate the contributor landscape almost entirely. This indicates that sustained and characteristic behaviors are disproportionately concentrated in strategically important parts of the codebase, while transient activity is largely confined to less critical areas.

Finding 1: *The distribution of developer types varies considerably across projects and becomes even more distinct when considering only the restricted set of returners and explorers. In critical files, the dominance of R&E is more pronounced, and the restricted set offers a clearer picture of the core contributors who maintain deep or broad engagement with the codebase.*

We also examined how returners, explorers, and tourists visited the codebase and whether there were “islands” of files exclusively touched by one category. For each project, we computed the sets of files visited by returners, explorers, and tourists, and then visualized their intersections using Venn diagrams. The results (see Figure 4) reveal considerable overlap across the three groups, indicating shared engagement with many parts of the codebase. However, in several projects, we also observed substantial non-overlapping regions, suggesting that each group contributes to distinct subsets of files. Notably, the returners’ “islands” in Cassandra, Flink, Hadoop, Hive, Ignite, and Kafka point to code specialization, while the isolated file regions, primarily associated with tourists and explorers, may correspond to onboarding activity or peripheral contributions.

Finding 2: *Although developers frequently interact with overlapping parts of the codebase, several projects present distinct file "islands" visited exclusively by returners, explorers, or tourists. This suggests specialization among returners and peripheral or onboarding behavior among tourists and explorers.*

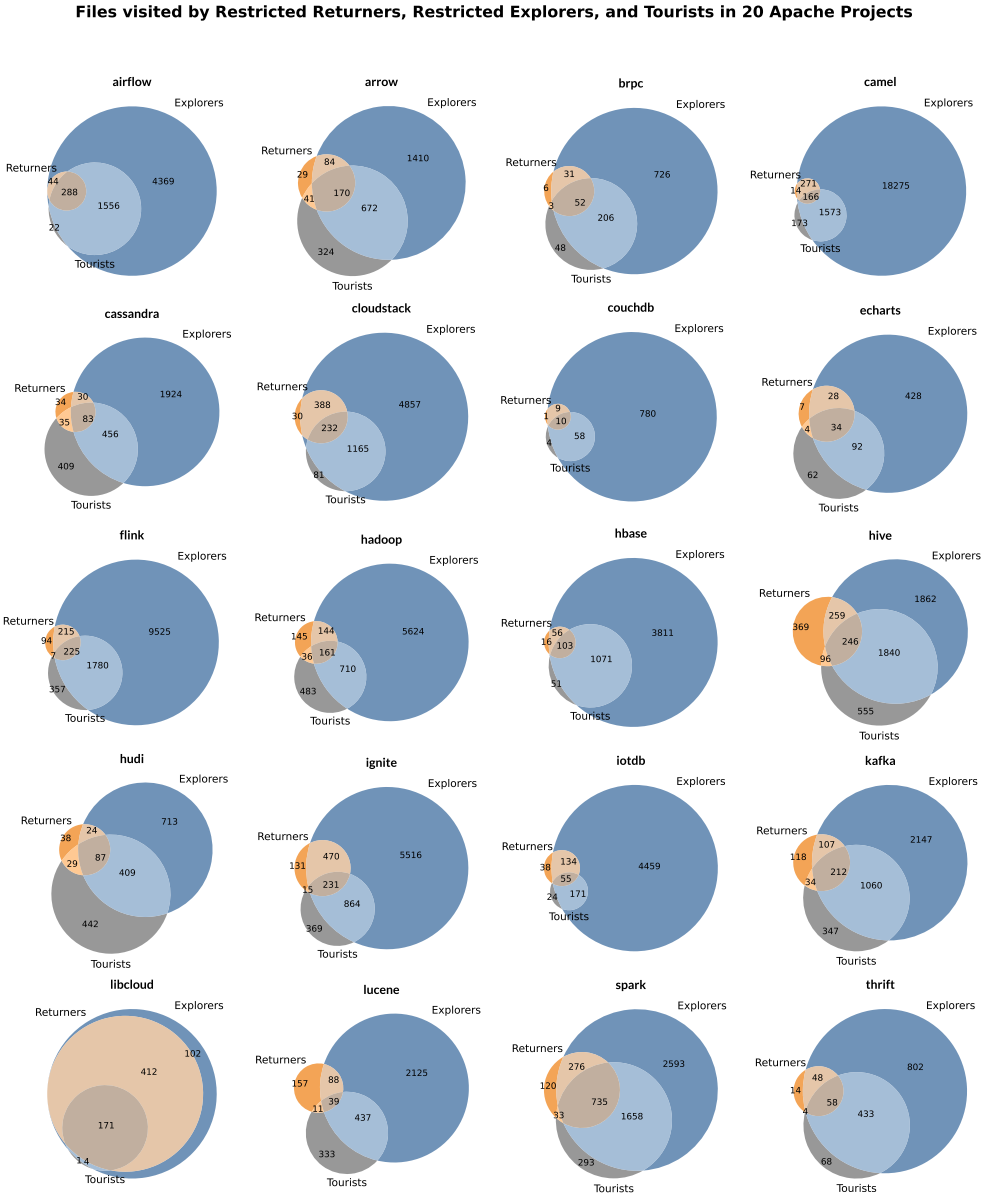
Figure 4 – File visit overlapping patterns among Returners (orange), Explorers (dark blue), and Tourists (gray), using the global classification.



Source: The author.

To assess whether these exclusive islands reflect persistent behavioral tendencies or are artifacts of borderline classifications, we applied the restricted classification (Figure 5). In some projects, such as Flink, Kafka, Hive, and Lucene, the exclusive islands observed in the global classification persisted almost unchanged, suggesting stable specialization where returners and explorers consistently focus on distinct subsets of files. In others, such as Airflow and CouchDB, these islands largely disappeared under the stricter thresholds, indicating that borderline or transient contributors drove the apparent specialization.

Figure 5 – File visit overlapping patterns among restricted Returners (orange), restricted Explorers (dark blue), and Tourists (gray).



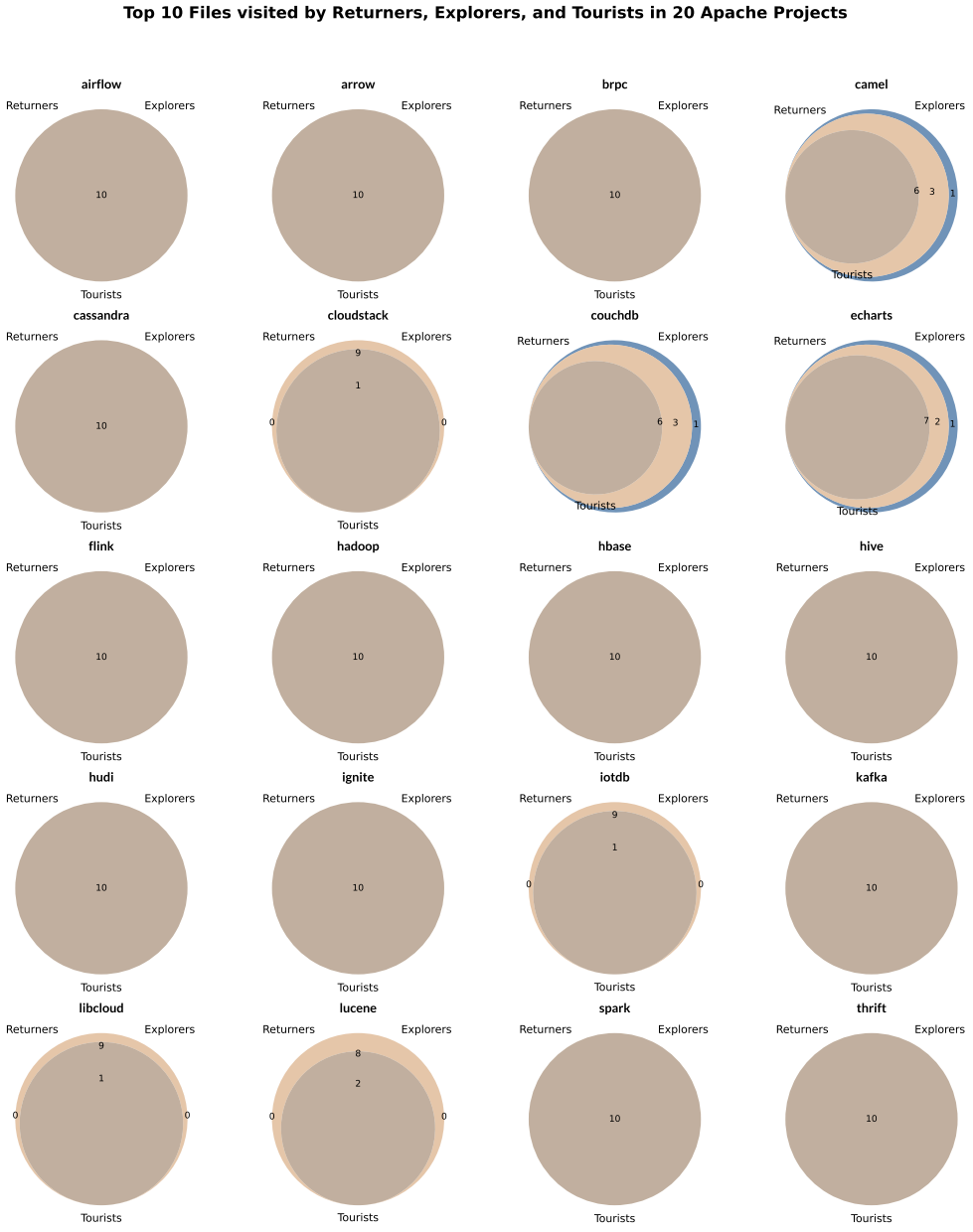
Source: The author.

Finding 3: *The restricted classification highlights robust specialization patterns by removing borderline cases. Projects where exclusive islands persist indicate stable and consistent role-based file ownership, while their disappearance suggests transient or less clearly defined behavioral tendencies.*

This observation becomes even more evident when narrowing the scope to the top-10 most critical files in each project (see Figure 6). Here, a noticeable behavioral shift occurs: returners and explorers now dominate these strategic areas, with tourists becoming visibly marginal. In 13 out of 20 projects, all groups interacted with all top-10 files, suggesting high collaboration in maintaining the most critical parts of the codebase. However, explorer-only

islands were still present in a few cases (e.g., Camel, CouchDB, and ECharts), indicating that even within critical areas, dispersed behavioral patterns persist.

Figure 6 – Overlapping patterns of file visits among Returners (orange), Explorers (dark blue), and Tourists (gray), restricted to the top-10 most critical files (global classification).

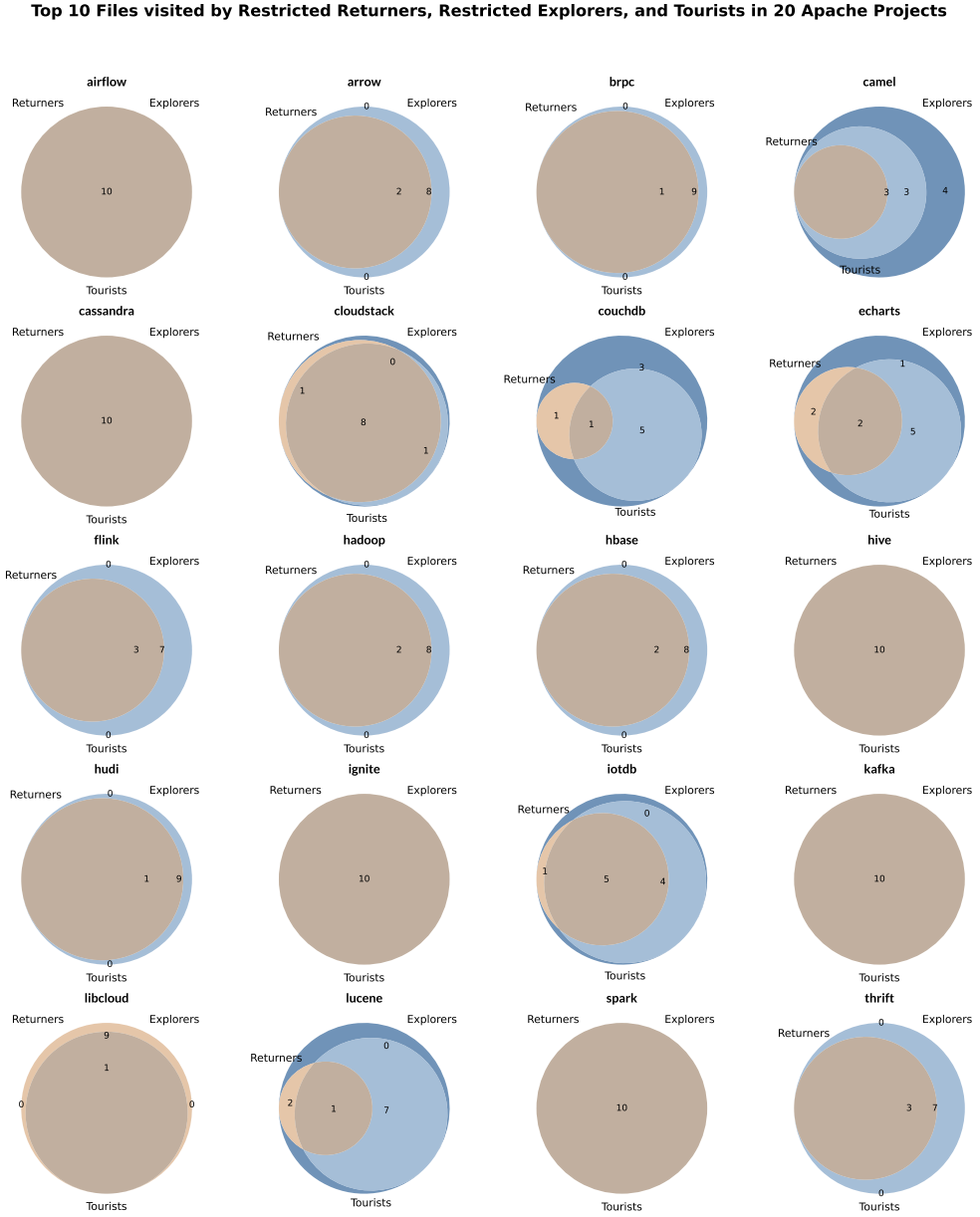


Source: The author.

When applying the restricted classification, these explorer-only islands not only persisted in some projects but also emerged in others (see Figure 7). This occurs because the stricter thresholds filter out borderline contributors, leaving a smaller but behaviorally consistent set of explorers whose work is concentrated in specific critical files. As a result, certain files that appeared shared in the global classification become exclusive to explorers in the restricted view,

revealing patterns of robust specialization that were previously masked by mixed contributions.

Figure 7 – Overlapping patterns of file visits among restricted Returners (orange), restricted Explorers (dark blue), and Tourists (gray), restricted to the top-10 most critical files.



Source: The author.

Finding 4: *In critical files, returners and explorers dominate, while tourists play a marginal role. The restricted classification reveals that, in some projects, explorers hold exclusive control over certain critical files, exposing robust specialization patterns that were masked in the global view.*

4.2 R&E and Code Knowledge (RQ2)

To address RQ2, we investigate to what extent the Returners and Explorers (R&E) framework can serve as a proxy for code knowledge in open-source projects. Specifically, we examine the relationship between developer categories (returners, explorers, tourists) and the Truck Factor (TF)—a proxy for system-level code ownership and critical developer expertise.

We computed the Truck Factor (TF) for each project following the methodology proposed by Avelino *et al.* (2016), using two distinct expertise metrics: Degree of Authorship (DOA) and Degree of Ownership Entropy (DOE). To identify expert developers, we adopted the threshold values recommended by the original study for open-source systems: $k = 0.7$ for both DOA and DOE scores, and a cumulative file coverage threshold of 0.5. This means that a developer is considered part of the TF if their expertise score exceeds 0.7 and their contributions collectively cover at least 50% of the source code files.

Table 5 – Truck Factor Metrics per Project (DOA vs. DOE).

Project	DOA-TF	DOA-TF-R	DOA-TF-E	DOA-TF-T	DOE-TF	DOE-TF-R	DOE-TF-E	DOE-TF-T
airflow	29	0	29	0	56	1	55	0
arrow	13	1	12	0	34	2	31	1
brpc	1	0	1	0	3	0	3	0
camel	8	0	8	0	20	0	20	0
cassandra	15	2	13	0	25	6	19	0
cloudstack	29	6	23	0	50	13	35	2
couchdb	5	0	5	0	8	0	8	0
echarts	2	0	2	0	3	0	3	0
flink	42	2	40	0	75	10	65	0
hadoop	39	3	36	0	55	8	47	0
hbase	20	4	16	0	48	6	42	0
hive	33	8	24	1	54	18	35	1
hudi	27	4	23	0	46	6	40	0
ignite	14	4	10	0	31	5	26	0
iotdb	17	2	15	0	26	3	23	0
kafka	44	9	35	0	88	20	68	0
libcloud	3	1	2	0	9	3	6	0
lucene	10	0	10	0	26	4	22	0
spark	57	6	51	0	177	46	127	4
thrift	13	1	12	0	33	4	28	1

Thresholds used in truck factor computation: DOA and DOE: $k = 0.7$, both used coverage threshold $= 0.5$.
Source: The author.

The results presented in Table 5 show that DOE-based TF values are consistently higher than DOA-based ones across all projects. This consistent difference reflects the stricter nature of the DOE metric, which captures ownership dispersion and tends to assign expert

status to developers who exhibit sustained engagement across complex or less redundant file contributions. In contrast, DOA may more easily assign expertise to concentrated authorship patterns, including contributors with fewer but impactful commits.

Across all projects, explorers constitute the majority of developers in the TF sets for both DOA and DOE. This pattern reflects the greedy nature of the TF computation strategy, which tends to favor developers who have modified a large range of files, a behavior that characterizes explorers. In contrast, returners, whose contributions are often concentrated in specific areas of the codebase, are less frequently selected by this strategy. While their depth of engagement may indicate specialization, it does not necessarily result in widespread coverage, which is a key factor in TF identification.

Although tourists are, by definition, marginal contributors, a very small number of them are occasionally included in TF sets, particularly under the DOE metric. These cases are rare and project-specific, and they typically correspond to developers whose limited contributions affected highly critical or low-redundancy files. Rather than indicating sustained expertise, such inclusions highlight a known limitation of activity-based metrics, which may attribute disproportionate importance to isolated but structurally impactful changes.

Finding 5: *Explorers dominate the Truck Factor (TF) sets under both DOA and DOE metrics, reflecting the preference of TF algorithms for breadth of contribution and wide file coverage. Returners appear less frequently due to the localized nature of their activity. Tourists are almost entirely absent from TF sets, with only rare exceptions, more common under DOE, linked to isolated contributions in highly critical or low-redundancy files.*

When applying the restricted R&E classification (Table 6), which retains only behaviorally prototypical returners and explorers, the overall composition of TF sets remains largely unchanged. As the restricted groups are strict subsets of the global classification, reductions in absolute counts are expected. However, the relative dominance of explorers persists under both DOA and DOE.

Importantly, the restricted classification makes explicit the presence of a residual group (Not-RE), composed of developers with sufficient activity to be evaluated but whose behavioral ratios fall between the returner and explorer extremes. A substantial fraction of TF members belongs to this intermediate group, indicating that TF-based expertise identification does not require strongly polarized behavioral profiles.

Table 6 – Truck Factor Metrics under Restricted Classification (DOA vs. DOE).

Project	DOA-TF	DOA-TF-R	DOA-TF-E	DOA-TF-T	DOA-TF-Not-RE	DOE-TF	DOE-TF-R	DOE-TF-E	DOE-TF-T	DOE-TF-Not-RE
airflow	29	0	19	0	10	56	0	31	0	25
arrow	13	0	2	0	11	34	0	10	1	23
brpc	1	0	1	0	0	3	0	2	0	1
camel	8	0	5	0	3	20	0	13	0	7
cassandra	15	0	3	0	12	25	0	4	0	21
cloudstack	29	0	8	0	21	50	1	11	2	36
couchdb	5	0	3	0	2	8	0	4	0	4
echarts	2	0	1	0	1	3	0	1	0	2
flink	42	0	10	0	32	75	0	13	0	62
hadoop	39	0	9	0	30	55	0	9	0	46
hbase	20	0	6	0	14	48	0	9	0	39
hive	33	1	6	1	25	54	1	7	1	45
hudi	27	0	2	0	25	46	0	3	0	43
ignite	14	0	6	0	8	31	0	9	0	22
iotdb	17	0	6	0	11	26	0	7	0	19
kafka	44	0	4	0	40	88	1	11	0	76
libcloud	3	1	1	0	1	9	1	2	0	6
lucene	10	0	1	0	9	26	0	3	0	23
spark	57	1	13	0	43	177	2	31	4	140
thrift	13	0	8	0	5	33	0	14	1	18

Source: The author.

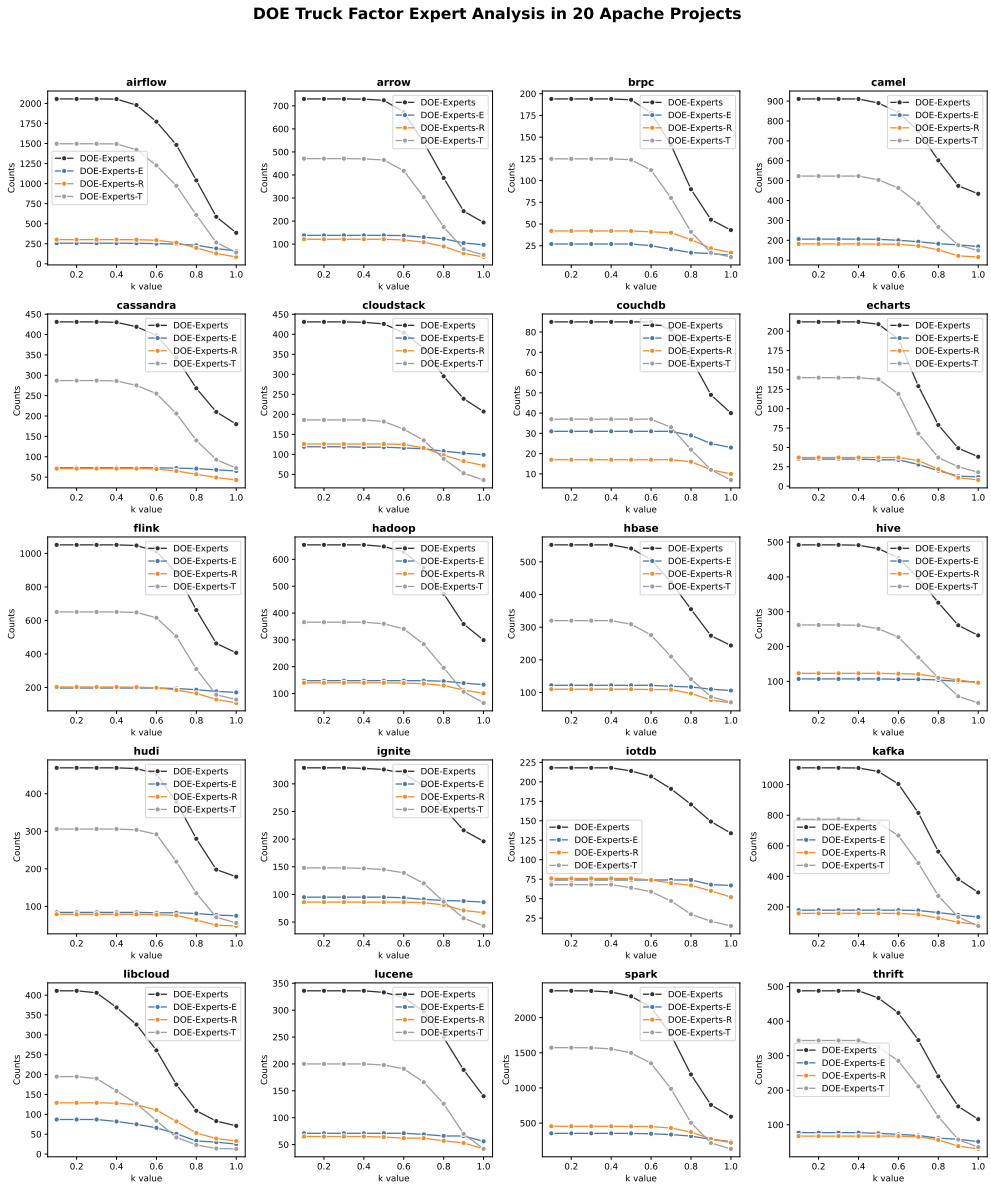
Finding 6: *Because the restricted R&E classification forms a strict subset of the global population, changes in absolute TF counts are expected. More importantly, even when considering only behaviorally prototypical developers, TF sets remain predominantly composed of explorers, while many highly specialized returners are excluded. This reinforces the conclusion that TF-based approaches prioritize breadth of contribution over depth, independently of behavioral classification strictness.*

To examine how sensitive expert identification is to threshold variation, we analyze how the composition of Truck Factor experts evolves as the parameter k changes. The following figures compare this behavior under DOE and DOA metrics, as well as under global and restricted R&E classifications.

To further assess the relationship between behavioral roles and technical expertise, we analyzed how the composition of DOE-based experts evolves as the threshold k varies. Figure 8 summarizes the results across all 20 projects, with k ranging from 0.1 to 1.0 (in increments of 0.1). For each value, we computed the DOE-based Truck Factor and categorized the resulting experts according to their R&E classification.

As the figure shows, the number of experts remains relatively stable at lower thresholds but tends to decline beyond a project-specific inflection point. This reduction disproportionately affects tourists, whose inclusion in the expert set is highly sensitive to the value of k. In contrast, returners and explorers display greater resilience, suggesting that their contributions consistently meet DOE-based expertise criteria across a wide range of thresholds.

Figure 8 – The influence of DOE-TF's *k* threshold on the composition of experts across R&E roles.



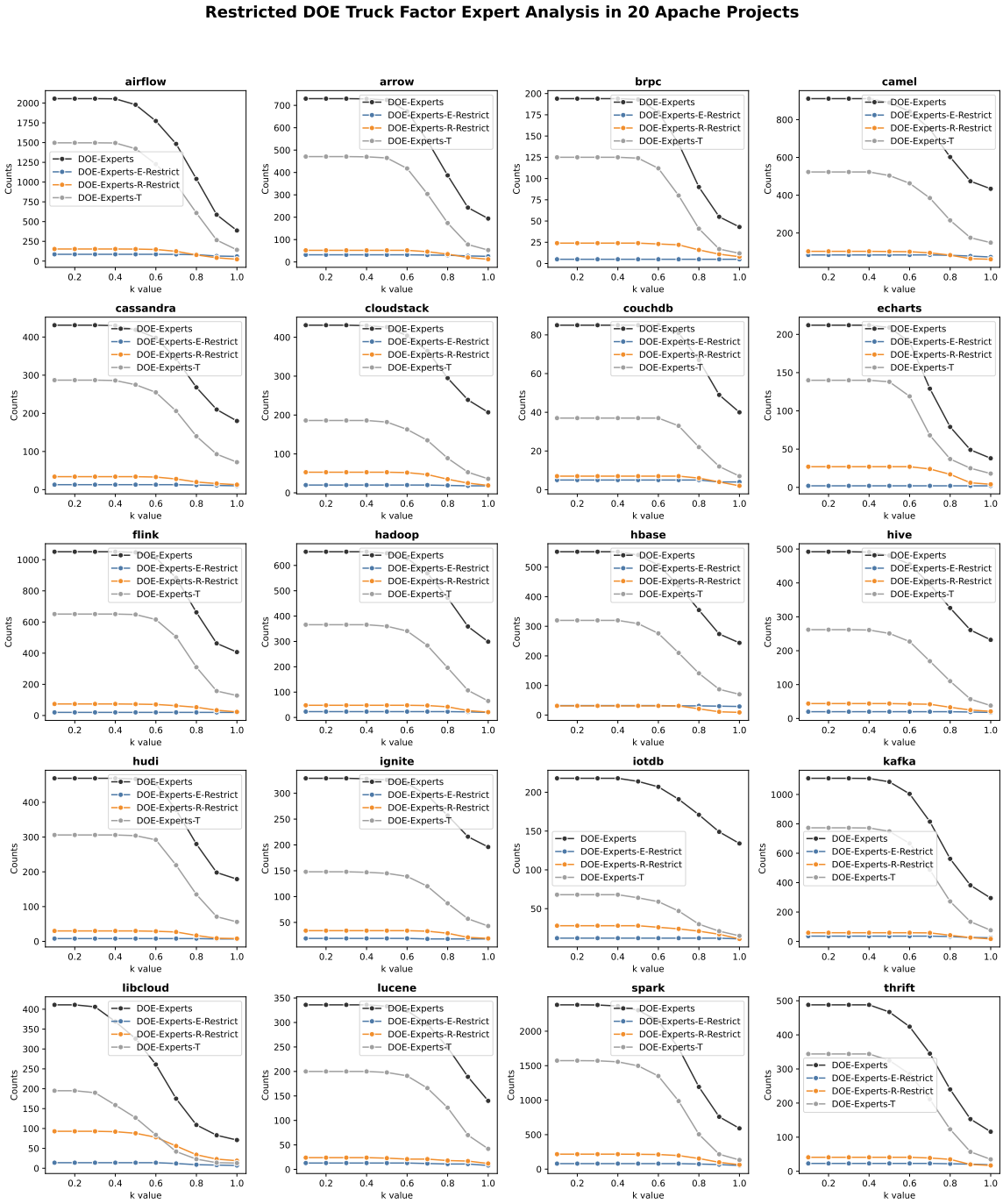
Source: The author.

To assess whether these trends persist when only strongly characteristic behavioral profiles are considered, we replicated the analysis using the **restricted classification** of returners and explorers, which filters out borderline developers and retains only those with more distinct behavioral patterns. Figure 9 presents the evolution of expert composition under this stricter criterion.

The restricted view confirms the robustness of returners and explorers as indicators of sustained code expertise. Even under stricter behavioral filters, these roles continue to dominate the expert set, while tourists remain largely excluded across all thresholds.

As a comparative baseline, we repeated the same analysis using the **DOA-based**

Figure 9 – Evolution of expert composition across k values using DOE-based Truck Factor (restricted classification).

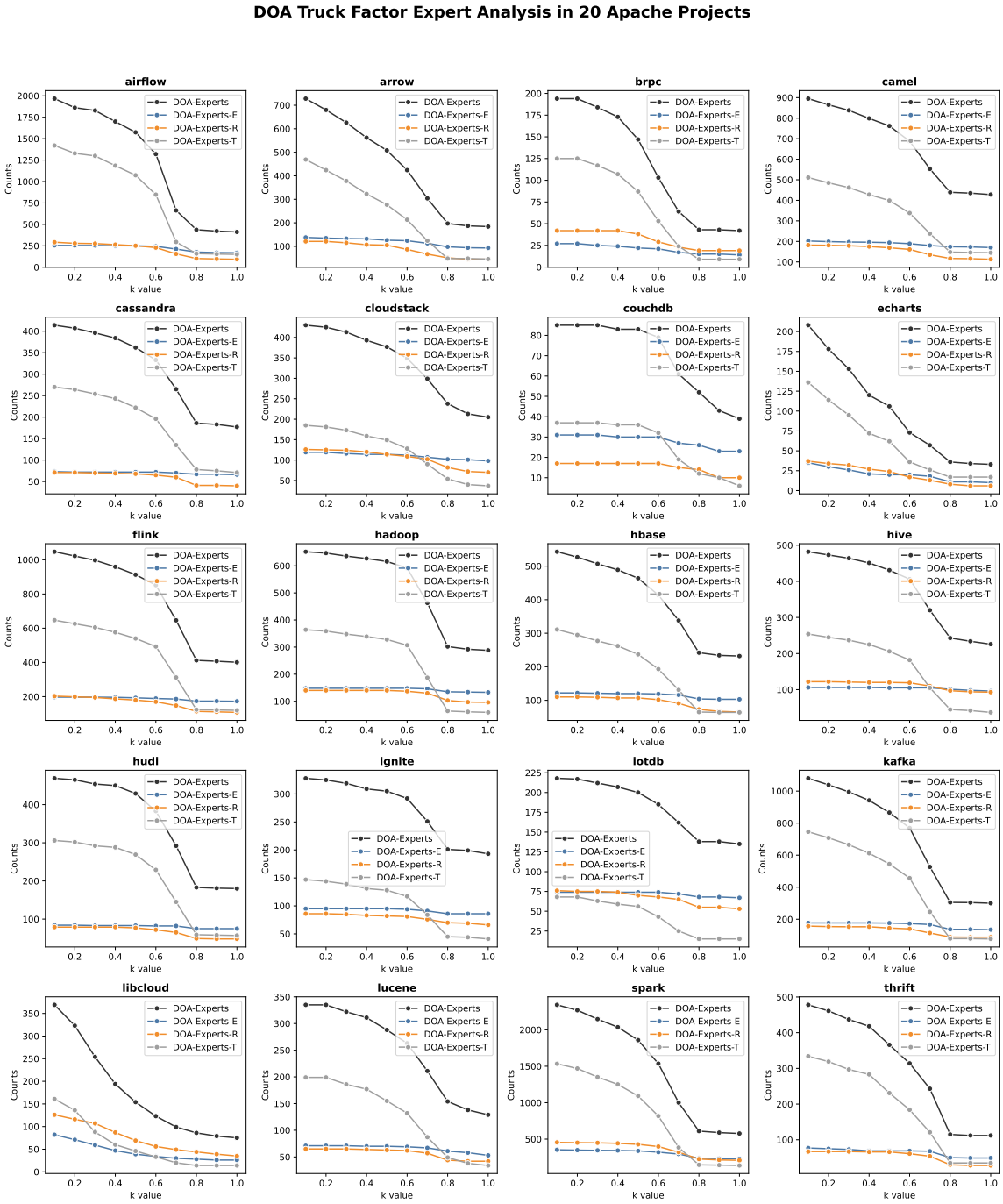


Source: The author.

Truck Factor, applying the standard global classification of behavioral roles. Figure 10 shows that the trends are consistent with those observed in DOE, although some variation exists in sensitivity to thresholding.

Together, these findings reinforce the conceptual alignment between long-term behavioral engagement and technical authority. Returners and explorers demonstrate consistent expertise profiles under both DOA and DOE, with and without classification restriction, whereas

Figure 10 – Evolution of expert composition across k values using DOA-based Truck Factor (global classification).



Source: The author.

tourists remain peripheral.

Finding 7: *As the DOE metric is inherently based on the distribution of file-level changes, developers with minimal and non-recurrent contributions are naturally filtered out as the expertise threshold increases. More importantly, despite variations in the DOE threshold, the relative presence of returners and explorers remains stable, even under the restricted classification. This stability indicates that R&E profiles capture behavioral properties that are robust to parameter variation in activity-based expertise metrics.*

To further validate the association between developer roles and code expertise, we analyze the overlap between Truck Factor experts and R&E behavioral roles using Venn diagrams. This analysis contrasts DOE- and DOA-based expert identification under both global and restricted classifications, highlighting how different expertise metrics capture behavioral specialization.

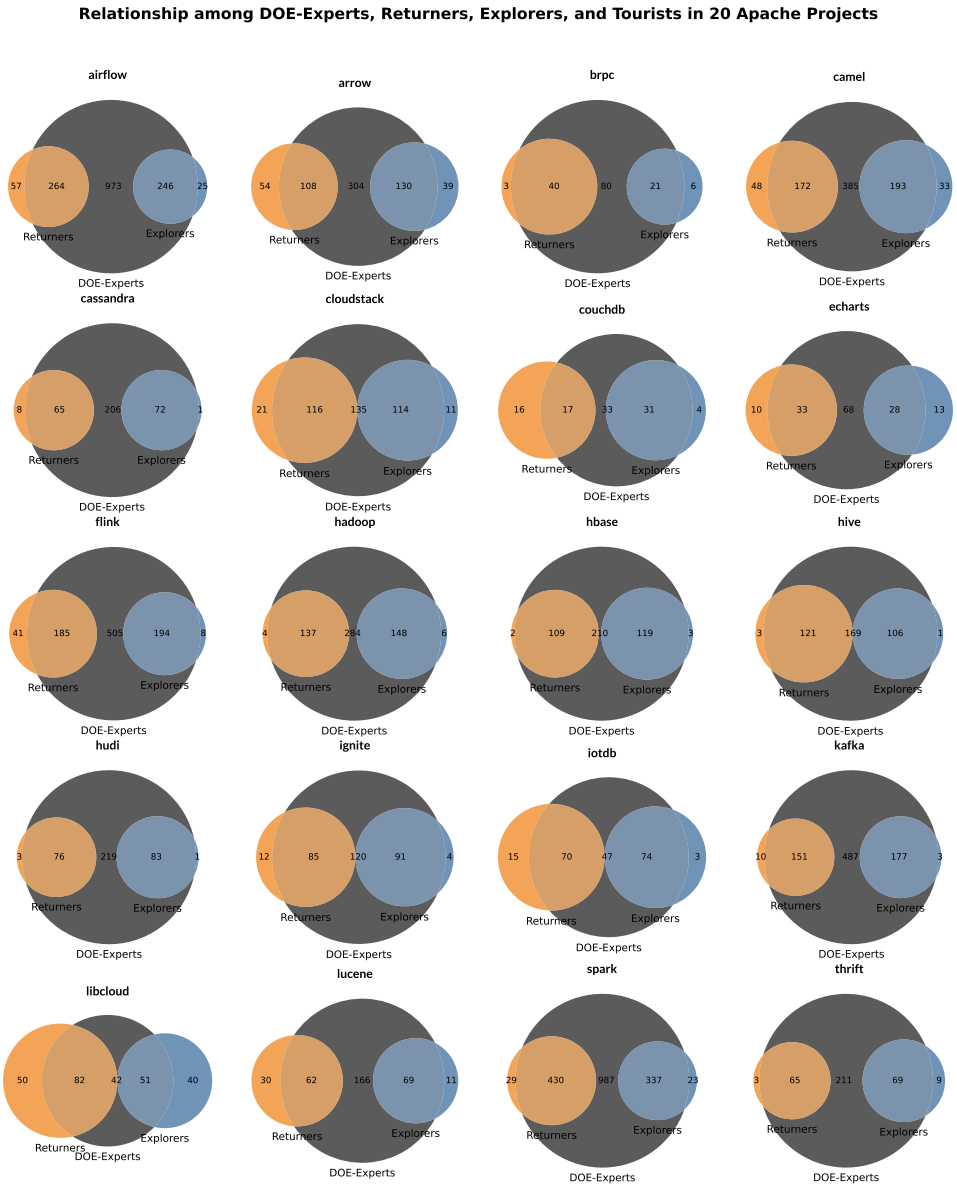
We first examine the overlap between DOE-based experts and R&E roles. Figure 11 presents the results under the global classification. A strong alignment emerges: most DOE-based experts are either returners or explorers, reinforcing the idea that these behavioral roles are closely tied to sustained code knowledge. However, a non-negligible portion of returners and explorers falls outside the expert set.

A closer inspection reveals two main causes for this partial mismatch. First, some developers contributed exclusively to files that were later removed, preventing DOE computation on the current version of the codebase. Second, some contributors working on live files scored below the DOE threshold. This threshold sensitivity is particularly evident in projects such as Airflow, where the number of identified experts begins to decline around $k = 0.7$ (Figure 8). In these cases, minor adjustments to the k parameter may increase expert recall without significantly affecting precision.

To assess whether this alignment strengthens when focusing on more characteristic behavioral profiles, we apply the restricted classification of returners and explorers. Figure 12 shows that the overlap becomes even more pronounced.

Under this stricter view, several projects, such as Ignite, IoTDB, and Kafka, exhibit near-complete overlap between restricted returners, explorers, and DOE-based experts. This improvement reflects the removal of borderline contributors, increasing precision at the cost of reduced coverage.

Figure 11 – Overlapping patterns among DOE-based experts (dark gray), Returners (orange), and Explorers (dark blue), using the global classification.

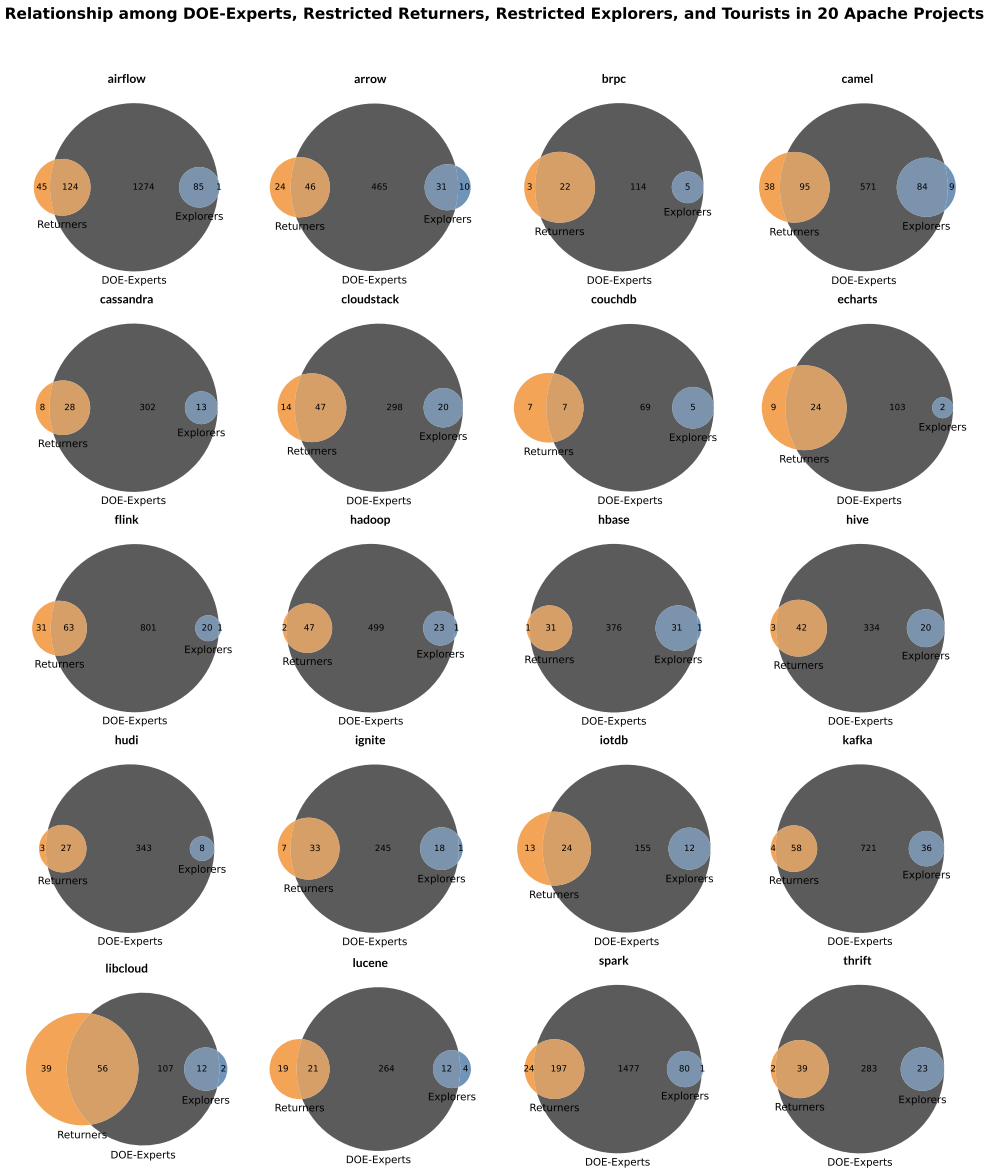


Source: The author.

Finding 8: *Because the DOE metric is based on the distribution of file-level changes, developers with minimal or non-recurrent contributions are inherently unlikely to be identified as experts. Consequently, tourists are structurally excluded from the DOE-based expert set by definition. Among developers with sufficient activity, however, a strong alignment emerges between DOE-based experts and the R&E profiles. This alignment persists under the restricted classification, indicating that returner and explorer behaviors capture properties consistent with structural code expertise, even when borderline cases are excluded.*

We now contrast these results with DOA-based expert identification. Unlike DOE, the overlap patterns observed under DOA differ substantially, particularly for returners.

Figure 12 – Overlapping patterns among DOE-based experts (dark gray), restricted Returners (orange), and restricted Explorers (dark blue).



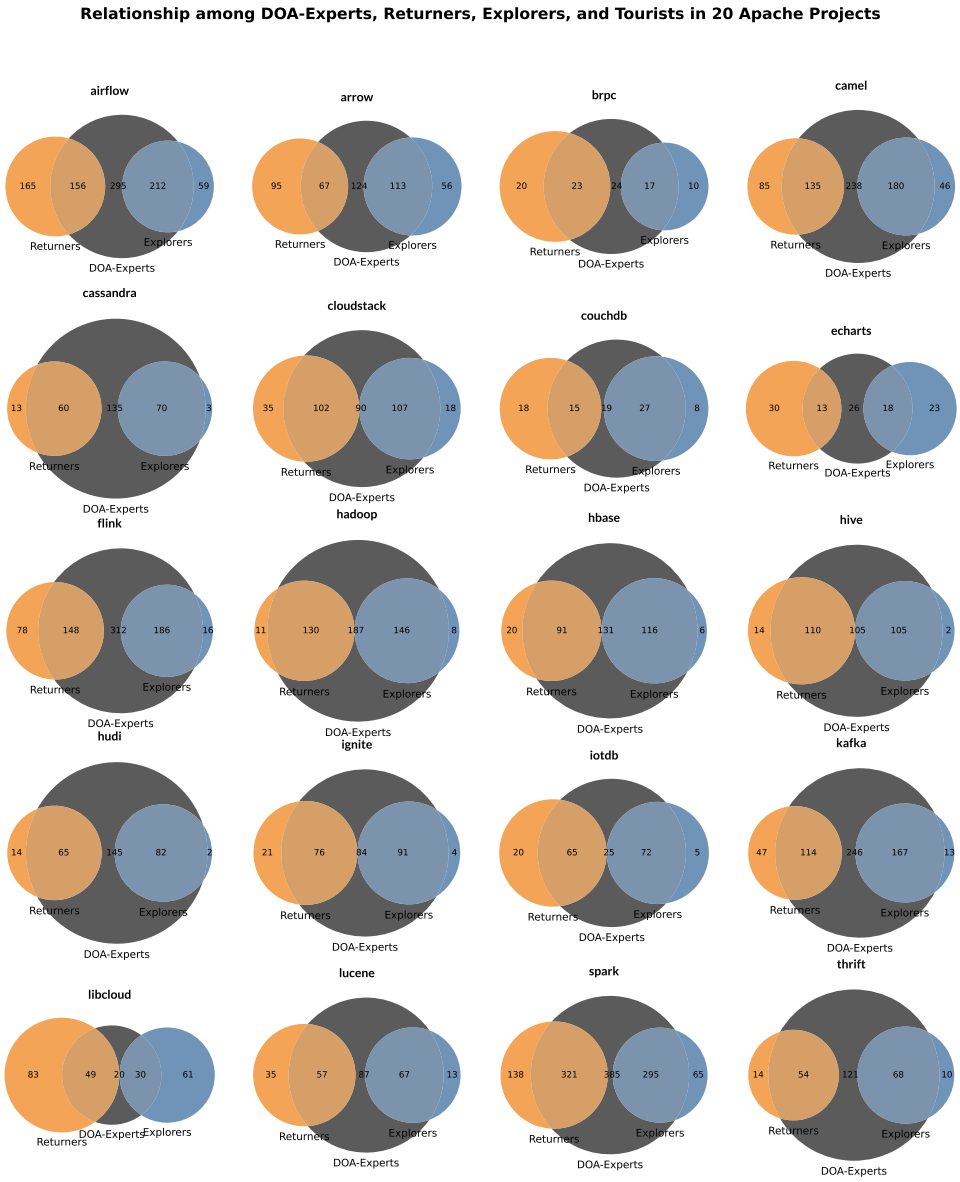
Source: The author.

In the global classification, large portions of returners remain outside the expert set, forming noticeable “islands” across projects such as *airflow*, *camel*, *hadoop*, and *ignite*. Although explorers tend to have greater inclusion, a substantial fraction still falls outside the DOA expert set.

Applying the restricted classification improves the alignment between explorers and DOA-based experts; however, even under these stricter conditions, many returners are still not recognized as experts.

This contrast with DOE-based results suggests that DOA’s emphasis on concentrated authorship may underrepresent developers who exhibit strong but localized specialization,

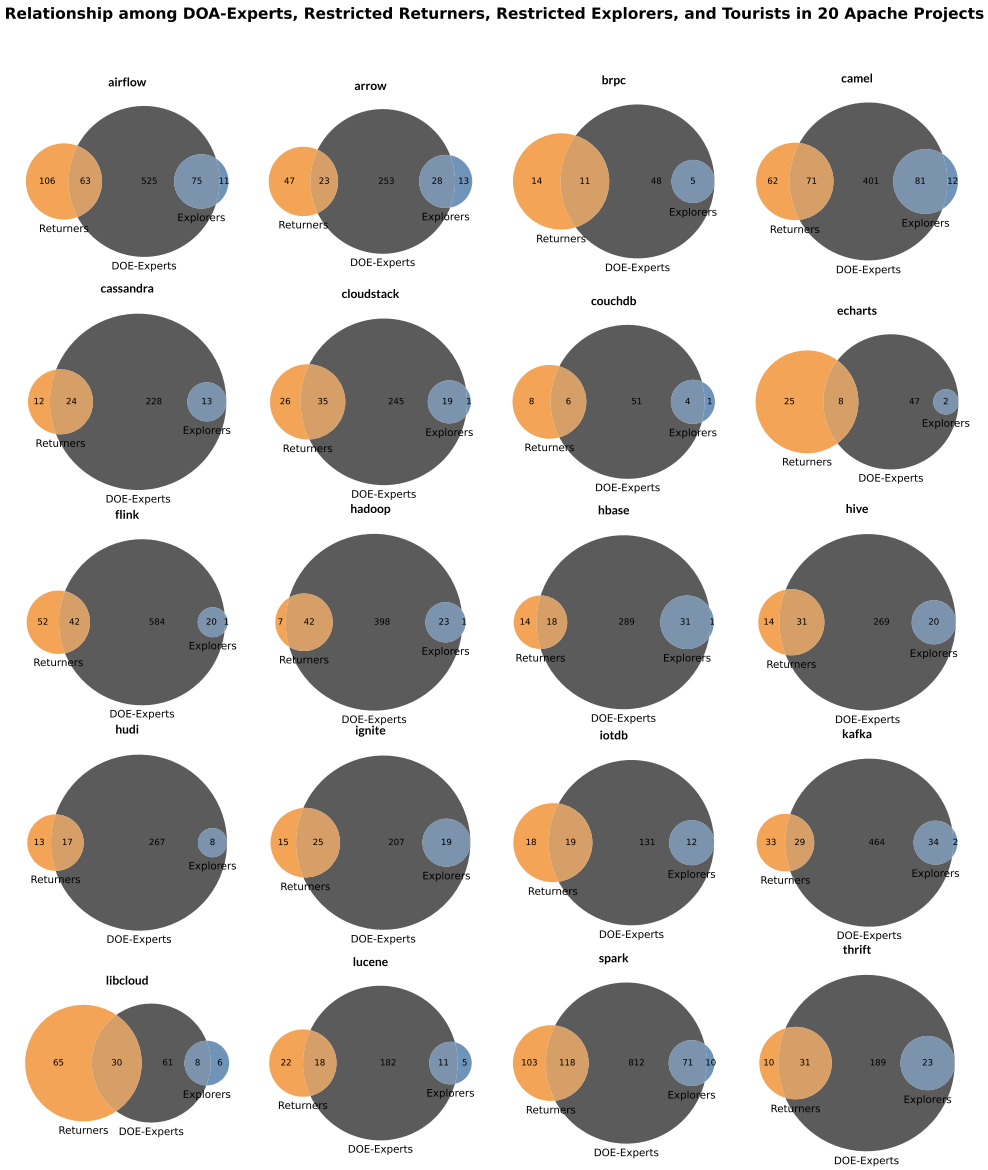
Figure 13 – Overlapping patterns among DOA-based experts (dark gray), Returners (orange), and Explorers (dark blue), using the global classification.



Source: The author.

particularly those focused on a small number of highly critical files.

Figure 14 – Overlapping patterns among DOA-based experts (dark gray), restricted Returners (orange), and restricted Explorers (dark blue).



Source: The author.

4.3 Discussion and Implications

This section reflects on the findings of our study and discusses their significance for software engineering research and practice. We begin by contrasting the behavioral insights provided by the Returners & Explorers (R&E) framework with traditional structural metrics such as the Truck Factor (TF), highlighting how behavioral classification reveals dimensions of knowledge distribution and project risk that are often overlooked (AVELINO *et al.*, 2016). We then examine the practical implications of these insights, outlining how the framework can support strategies for team organization, knowledge management, and long-term project sustainability (SOUSA *et al.*, 2018; SADOWSKI *et al.*, 2018). Finally, we consider

the applicability of our approach beyond software engineering, suggesting potential avenues for interdisciplinary research in domains that involve structured exploration and knowledge navigation (GOLDENBERG *et al.*, 2009).

4.3.1 Beyond Traditional Metrics: Behavioral Perspectives on Code Knowledge

The adaptation of the Returners & Explorers (R&E) framework to the software engineering domain uncovers behavioral dimensions that remain largely invisible to traditional metrics such as the Truck Factor (TF) (AVELINO *et al.*, 2016). While TF quantifies knowledge concentration by identifying the minimal subset of developers whose departure would compromise the system's maintainability, it treats contributors as homogeneous agents, prioritizing ownership over behavioral patterns. Our findings challenge this perspective by revealing that developers differ not only in what they know, but in how they navigate, specialize, and interact with the codebase (CATOLINO *et al.*, 2019).

The three-tier classification introduced by the R&E framework, returners, explorers, and tourists, illuminates these behavioral distinctions. Returners exhibit focused, repeated engagement with a limited set of files, suggesting deep, localized expertise (SOUSA *et al.*, 2018). Explorers, on the other hand, contribute across a broader range of components, supporting knowledge diffusion and systemic awareness. Tourists, despite representing a large portion of the contributor base (up to 76% in our dataset), are invisible to TF metrics due to their limited engagement, yet their presence reflects the permeability and onboarding dynamics of open-source ecosystems (RIGBY; BIRD, 2013).

Our empirical analysis shows that R&E profiles provide complementary insights to TF. For example, systems anchored by returners may display low TF scores, yet be vulnerable to the loss of specialized, irreplaceable knowledge. Conversely, projects dominated by explorers may exhibit higher TF values, yet possess greater resilience due to the redundancy of distributed knowledge. Thus, two systems with identical TF values may entail fundamentally different knowledge risk profiles, an insight only accessible through behavioral analysis.

This behavioral lens opens new avenues for project management. Critical modules may benefit from stable returners capable of maintaining architectural integrity, while peripheral or rapidly evolving components may be better suited to the agility of explorers. Recognizing this distinction enables more nuanced decisions around task allocation, contributor onboarding, and long-term sustainability planning (SADOWSKI *et al.*, 2018).

4.3.2 Implications for Software Engineering and Beyond

The R&E framework offers several practical and theoretical implications for software engineering research and practice. From a managerial standpoint, behavioral profiling provides a more granular understanding of expertise distribution. Organizations can monitor when essential modules become overly dependent on returners, signaling potential knowledge silos, or when exploratory behavior becomes dominant, indicating possible gaps in specialization. This complements traditional metrics such as TF by focusing on the developer's interaction style rather than just their cumulative impact (SOUSA *et al.*, 2018).

These insights are directly applicable to strategies involving team composition, succession planning, and knowledge transfer. By balancing returners and explorers across critical system components, managers can promote both continuity and adaptability, ensuring robustness against turnover while preserving deep technical expertise.

Beyond software engineering, our study exemplifies the broader applicability of mobility-inspired frameworks to complex digital systems. The use of geodesic distance as a structural proxy for spatial dispersion enables behavioral modeling in hierarchical information spaces, ranging from scientific collaboration graphs and knowledge repositories to corporate intranets and educational platforms (GOLDENBERG *et al.*, 2009).

Conversely, insights from the software domain may enrich the original mobility literature (PAPPALARDO *et al.*, 2015). Unlike urban mobility, where individuals navigate static environments, software developers engage in co-evolutionary dynamics: they not only traverse the system but also reshape it. Studying returner-explorer transitions under these conditions offers a unique perspective on how navigation behaviors interact with structural evolution. Moreover, the availability of rich version-control data enables temporal analyses rarely achievable in traditional mobility studies, allowing researchers to investigate how developers switch roles over time, how knowledge accumulates, and how collaboration patterns evolve (MENS *et al.*, 2005).

In sum, the adaptation of the R&E framework expands the analytical repertoire available for understanding knowledge distribution in software projects, offering a richer, behaviorally grounded complement to existing structural metrics and opening new possibilities for interdisciplinary research across digital and physical mobility contexts.

4.4 Summary

This chapter presented and discussed the empirical results of applying the Returners & Explorers (R&E) framework to twenty large-scale Apache projects. We first analyzed the distribution of returners, explorers, and tourists under both the global and restricted classifications. The results showed that tourists dominate the contributor base in most projects, while returners and explorers, though fewer, are essential to the maintenance of critical files. The restricted classification further highlighted stable specialization patterns, filtering out transient contributors and revealing role-based file ownership.

We then investigated the relationship between R&E roles and knowledge-oriented metrics, particularly the Truck Factor (TF). Explorers emerged as the majority within TF sets due to their broad engagement with the codebase, whereas returners, despite their localized expertise, were less frequently captured. This revealed an important limitation of TF metrics: their reliance on breadth over depth. Restricted classifications reinforced these insights, demonstrating the robustness of R&E as a behavioral proxy for developer expertise.

Finally, the discussion emphasized that R&E complements traditional structural measures by exposing behavioral dynamics of knowledge distribution. Returners contribute depth and specialization, explorers ensure breadth and redundancy, and tourists reflect project permeability and onboarding.

5 CONCLUSION AND FUTURE WORK

This chapter summarizes the main findings of this dissertation and discusses their implications for understanding and modeling developer behavior in software engineering, particularly through the adaptation of the Returners and Explorers framework to source code analysis. By integrating behavioral classifications with expertise metrics such as DOA, DOE, and the Truck Factor, this study has provided new insights into how different contribution patterns relate to code knowledge distribution and critical file maintenance. The results not only validate the applicability of the adapted framework to large-scale open-source projects but also highlight its potential as a complementary approach to existing expertise identification methods. Finally, we outline directions for future research that can expand the scope, improve the precision, and explore additional applications of this approach.

5.1 Conclusion

This study proposed a novel adaptation of the Returners and Explorers (R&E) framework originally conceived to characterize human mobility patterns to the domain of software engineering. By modeling code modifications as navigational behavior and using geodesic distance over the directory structure to quantify spatial dispersion, we classified developers into three behavioral profiles: *returners* (localized and specialized), *explorers* (broad and dispersed), and *tourists* (transient and episodic).

The application of this behavioral lens to 20 long-lived, high-impact projects from the Apache Software Foundation revealed several important findings. Although tourists make up the majority of contributors, it is the returners and explorers who carry out most of the modifications in the system's critical regions, highlighting their essential role in maintaining complex and high-impact code. While file visitation patterns often overlap between roles, some projects exhibit clear behavioral "islands," with clusters of files exclusively modified by a single role, indicating specialization and differentiated responsibilities within the development process.

The analysis also showed that explorers appear more frequently in Truck Factor (TF) developer sets, a result consistent with the greedy file-based heuristics of TF computation, which tend to prioritize breadth of file coverage over recurrence of contributions. Furthermore, both returners and explorers are consistently recognized as code experts under TF and DOE-based models, reinforcing the R&E framework as a meaningful and behaviorally grounded proxy for

code knowledge.

These results reveal that the R&E framework offers a complementary and behaviorally nuanced perspective to traditional structural metrics such as Truck Factor. While TF identifies who holds critical knowledge, the R&E model explains *how* that knowledge is acquired, concentrated, or diffused—uncovering the invisible dynamics behind software evolution, contributor engagement, and knowledge sustainability.

5.2 Future Work

Building upon the contributions of this study, several promising directions emerge for future research. One avenue is the longitudinal tracking of behavioral transitions, examining how developers move between roles, such as from tourist to returner, throughout their involvement in a project, and how these shifts relate to onboarding processes, role formalization, or major refactorings. Another direction involves exploring role–task associations, investigating whether specific developer profiles correlate with distinct types of contributions; for example, assessing whether returners predominantly handle bug fixes while explorers are more engaged in new feature development, by integrating evidence from issue trackers and pull requests.

Finally, evaluating the framework in industrial and large-scale legacy systems—contexts often characterized by knowledge silos and higher turnover risks—can help determine its applicability and practical value beyond open-source environments.

5.3 Final Remarks

This work demonstrates the value of importing behavioral models from other domains, such as human mobility, into software engineering. By shifting the focus from static ownership metrics to dynamic interaction patterns, the Returners & Explorers framework enables a deeper understanding of how knowledge flows, accumulates, and is sustained within software systems.

As systems grow in complexity and contributor bases become more fluid, tools that capture not only who knows what, but also *how* they navigate and engage with code, will become increasingly vital. The R&E framework offers one such tool—grounded in empirical evidence, behaviorally interpretable, and extensible to a variety of contexts.

We hope this study paves the way for future efforts that further integrate behavioral

perspectives into the study of software development, fostering more resilient, sustainable, and knowledge-aware engineering practices.

REFERENCES

ALESSANDRETTI, L.; SAPIEZYNSKI, P.; SEKARA, V.; LEHMANN, S.; BARONCHELLI, A. Evidence for a conserved quantity in human mobility. **Nature Human Behaviour**, v. 2, n. 7, p. 485–491, 2018.

AVELINO, G.; PASSOS, L.; HORA, A.; VALENTE, M. T. A novel approach for estimating truck factors. In: **2016 IEEE 24th International Conference on Program Comprehension (ICPC)**. [S.l.: s.n.], 2016. p. 1–10.

BARBOSA, H.; BARTHELEMY, M.; GHOSHAL, G.; JAMES, C. R.; LENORMAND, M.; LOUAIL, T.; MENEZES, R.; RAMASCO, J. J.; SIMINI, F.; TOMASINI, M. Human mobility: models and applications. **Physics Reports**, v. 734, p. 1–74, 2018.

CATOLINO, G.; PALOMBA, F.; TAMBURRI, D. A.; SEREBRENIK, A.; FERRUCCI, F. Gender diversity and women in software teams: how do they affect community smells? In: **Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society**. [S.l.: s.n.], 2019. p. 11–20.

CURY, O.; AVELINO, G.; NETO, P. S.; VALENTE, M. T.; BRITTO, R. Source code expert identification: models and application. **Information and Software Technology**, v. 170, n. C, 2024.

FERREIRA, M.; MOMBACH, T.; VALENTE, M. T.; FERREIRA, K. Algorithms for estimating truck factors: a comparative study. **Software Quality Journal**, v. 27, n. 4, p. 1583–1617, 2019.

FRITZ, T.; MURPHY, G. C.; MURPHY-HILL, E.; OU, J.; HILL, E. Degree-of-knowledge: Modeling a developer's knowledge of code. **ACM Transactions on Software Engineering and Methodology**, v. 23, n. 2, 2014.

GOLDENBERG, J.; HAN, S.; LEHMANN, D. R.; HONG, J. W. The role of hubs in the adoption process. **Journal of Marketing**, v. 73, n. 2, p. 1–13, 2009.

GONZÁLEZ, M. C.; HIDALGO, C. A.; BARABÁSI, A.-L. Understanding individual human mobility patterns. **Nature**, v. 453, n. 7196, p. 779–782, 2008.

LUCAS, E. M.; OLIVEIRA, T. C.; SCHNEIDER, D.; ALENCAR, P. S. C. Knowledge-oriented models based on developer-artifact and developer-developer interactions. **IEEE Access**, v. 8, p. 218702–218719, 2020.

MENS, T.; WERMELINGER, M.; DUCASSE, S.; DEMEYER, S.; HIRSCHFELD, R.; JAZAYERI, M. Challenges in software evolution. In: **Eighth International Workshop on Principles of Software Evolution (IWPSE'05)**. [S.l.: s.n.], 2005. p. 13–22.

MONTANDON, J. a. E.; SILVA, L. L.; VALENTE, M. T. Identifying experts in software libraries and frameworks among github users. In: **Proceedings of the 16th International Conference on Mining Software Repositories**. IEEE Press, 2019. (MSR '19), p. 276–287. Available at: https://doi.org/10.1109/MSR.2019.00054.

PAPPALARDO, L.; SIMINI, F.; RINZIVILLO, S.; PEDRESCHI, D.; GIANNOTTI, F.; BARABÁSI, A. Returners and explorers dichotomy in human mobility. **Nature Communications**, v. 6, 2015.

RIGBY, P. C.; BIRD, C. Convergent contemporary software peer review practices. In: **Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2013. (ESEC/FSE 2013), p. 202–212. ISBN 9781450322379. Available at: https://doi.org/10.1145/2491411.2491444.

SADOWSKI, C.; SöDERBERG, E.; CHURCH, L.; SIPKO, M.; BACCHELLI, A. Modern code review: a case study at google. In: **Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice**. New York, NY, USA: Association for Computing Machinery, 2018. (ICSE-SEIP '18), p. 181–190. ISBN 9781450356596. Available at: https://doi.org/10.1145/3183519.3183525.

SCHLÄPFER, M.; DONG, L.; O'KEEFFE, K.; SANTI, P.; SZELL, M.; SALAT, H.; ANKLESARIA, S.; VAZIFEH, M.; RATTI, C.; WEST, G. B. The universal visitation law of human mobility. **Nature**, v. 593, n. 7860, p. 522–527, 2021.

SIMINI, F.; GONZÁLEZ, M. C.; MARITAN, A.; BARABÁSI, A.-L. A universal model for mobility and migration patterns. **Nature**, v. 484, n. 7392, p. 96–100, 2012.

SONG, C.; QU, Z.; BLUMM, N.; BARABÁSI, A.-L. Limits of predictability in human mobility. **Science**, v. 327, n. 5968, p. 1018–1021, 2010.

SOUSA, A.; ROCHA, L.; BRITTO, R.; AVELINO, G. An automated approach to identify source code files affected by architectural technical debt. In: PFAHL, D.; HUERTA, J. G.; KLÜNDER, J.; ANWAR, H. (Ed.). **Product-Focused Software Process Improvement. Industry-, Workshop-, and Doctoral Symposium Papers**. [S.l.: s.n.], 2025. p. 100–115.

SOUSA, L.; OLIVEIRA, A.; OIZUMI, W.; BARBOSA, S.; GARCIA, A.; LEE, J.; KALINOWSKI, M.; MELLO, R. de; FONSECA, B.; OLIVEIRA, R.; LUCENA, C.; PAES, R. Identifying design problems in the source code: a grounded theory. In: **Proceedings of the 40th International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2018. (ICSE '18), p. 921–931. ISBN 9781450356381. Available at: https://doi.org/10.1145/3180155.3180239.

SPADINI, D.; ANICHE, M.; BACCHELLI, A. Pydriller: python framework for mining software repositories. In: **Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. [S.l.: s.n.], 2018. p. 908–911.

WANG, C.; LI, Y.; CHEN, L.; HUANG, W.; ZHOU, Y.; XU, B. Examining the effects of developer familiarity on bug fixing. **Journal of Systems and Software**, v. 169, p. 110667, 2020.

WESOLOWSKI, A.; EAGLE, N.; TATEM, A. J.; SMITH, D. L.; NOOR, A. M.; SNOW, R. W.; BUCKEE, C. O. Quantifying the impact of human mobility on malaria. **Science**, v. 338, n. 6104, p. 267–270, 2012.

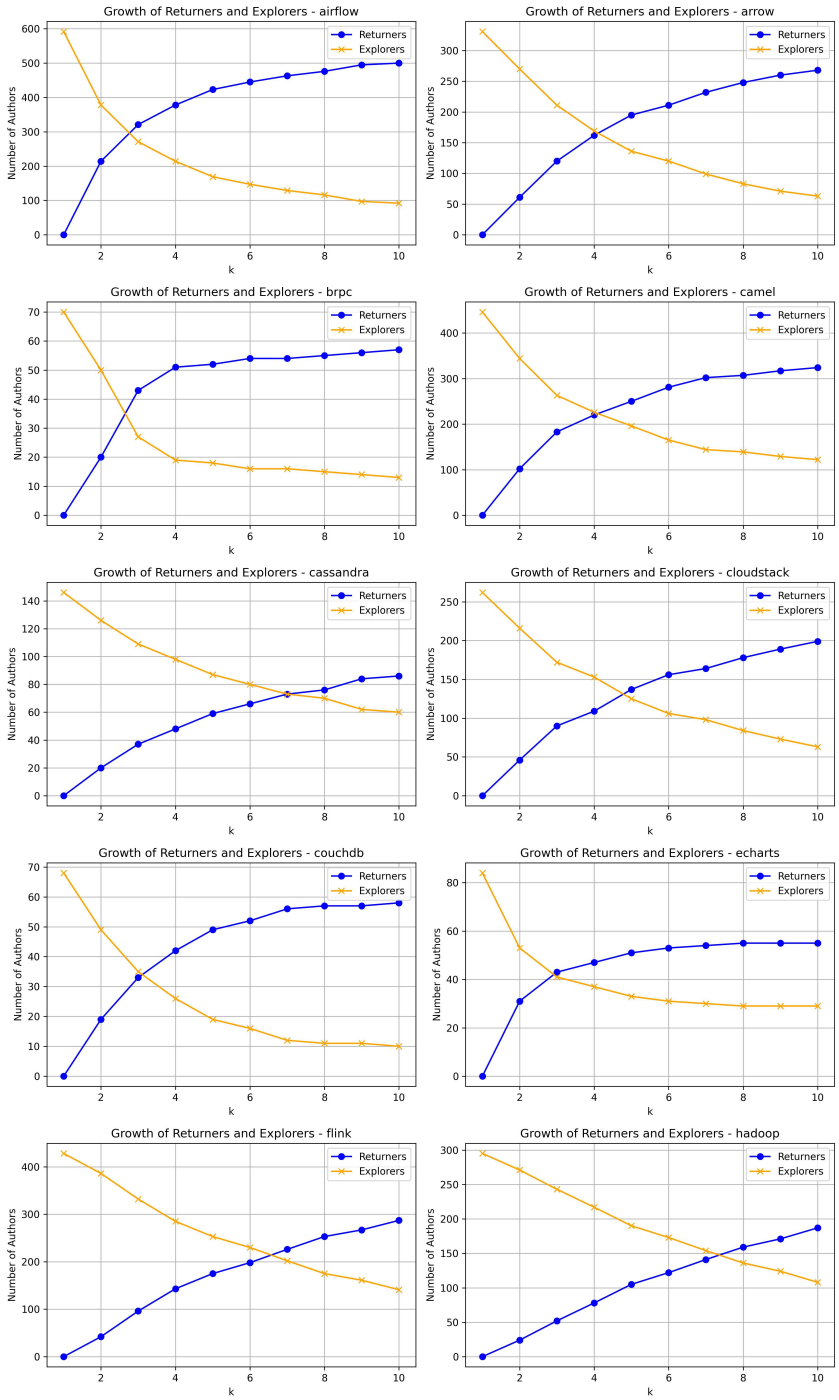
XU, Y.; XUE, J.; PARK, S.; YUE, Y. Towards a multidimensional view of tourist mobility patterns in cities: a mobile phone data perspective. **Computers, Environment and Urban Systems**, v. 86, p. 101593, 2021.

YAMASHITA, K.; MCINTOSH, S.; KAMEI, Y.; HASSAN, A. E.; UBAYASHI, N. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In: **Proceedings of the 14th International Workshop on Principles of Software Evolution**. New York, NY, USA: Association for Computing Machinery, 2015. (IWPSE 2015), p. 46–55. ISBN 9781450338165. Available at: https://doi.org/10.1145/2804360.2804366.

APPENDIX A – EVOLUTION OF RETURNERS AND EXPLORERS ACROSS *k*

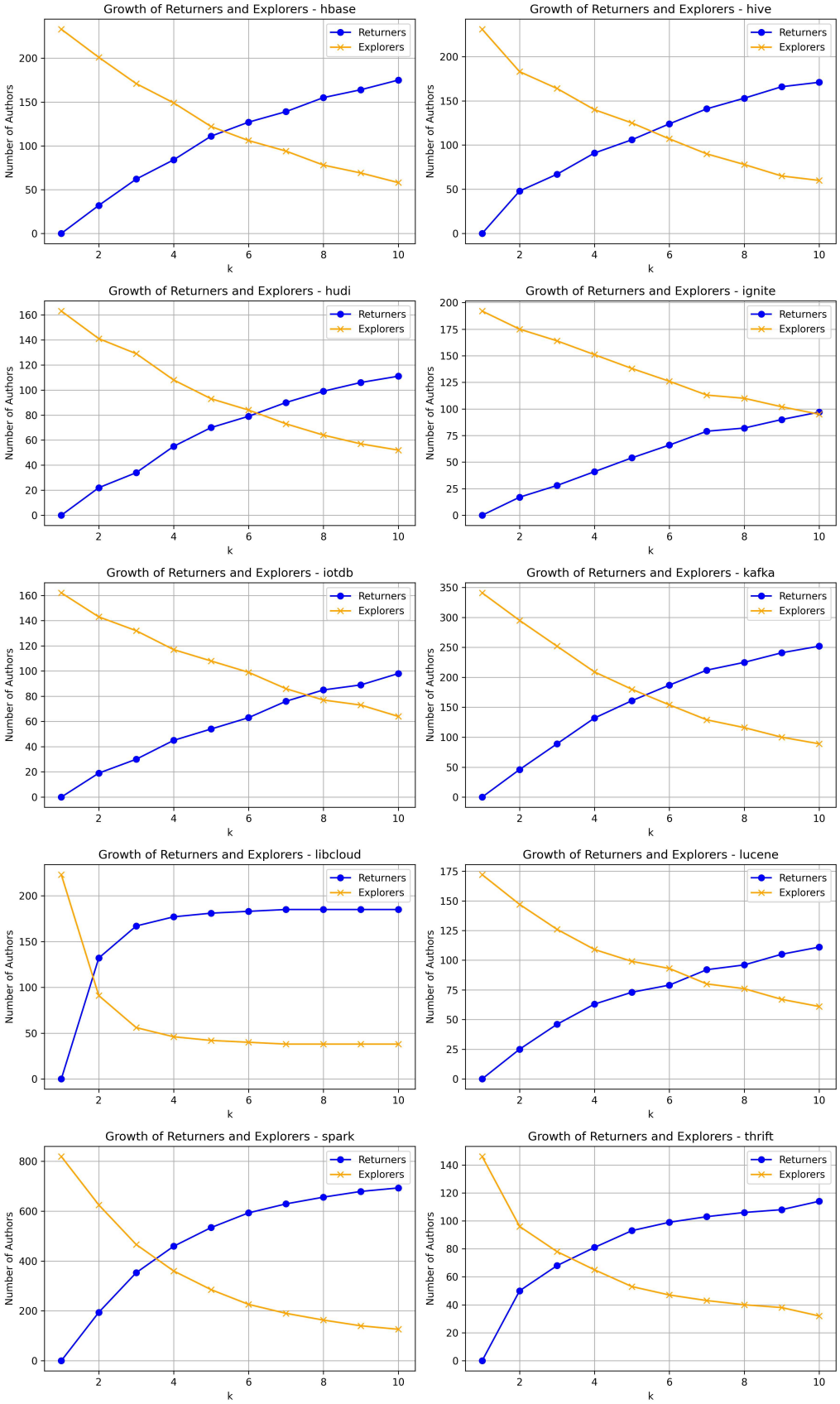
This appendix provides the detailed per-project evolution of returners and explorers as the value of *k* increases. These figures complement the analysis presented in Chapter 4.1, illustrating the stability of the observed trends across all analyzed projects.

Figure 1 – Evolution of Returners and Explorers across values of *k* for the first 10 Apache projects.



Source: The author.

Figure 2 – Evolution of Returners and Explorers across values of k for the remaining 10 Apache projects.



Source: The author.