A Handbook for InnerSource Contribution Functions

BACHELOR THESIS

Abderrahmane Bennani

Submitted on 2 September 2024

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Julian Hirsch, M. Sc.
Prof. Dr. Dirk Riehle, M.B.A.

Friedrich-Alexander-Universität
Faculty of Engineering

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Nuremberg, 2 September 2024

License

Nuremberg, 2 September 2024

Abstract

InnerSource refers to the adoption of open source-like practices and culture within the confines of an organization. As interest in InnerSource practices and adoption thereof grow, so do the challenges associated with them. It is important to accurately valuate InnerSource contributions to ensure the tax compliance of the organization. Especially when said contributions originate from branches located in different countries. In this paper, we investigate the characteristics that define an InnerSource contribution, as well as explore the existing classification systems in the literature. Building on this foundation, we develop a new classification system based on Semantic Versioning. This system is designed to facilitate the accurate valuation of InnerSource contributions by categorizing them according to OECD's DEMPE functions. We then apply said system on different contribution types and look at tools that can help facilitate and automate the use and adoption of the classification system. Finally, to ensure the system's validity, we outline a validation strategy that includes pilot testing and performance metrics.

Contents

Acronyms

ISSD Inner Source Software Development

DEMPE Development, Enhancement, Maintenance, Protection and Exploitation

OECD Organisation for Economic Co-operation and Development

NLP Natural Language Processing

SemVer Semantic Versioning 2.0.0

LLMs Large Pre-trained Models

1 Introduction

1.1 Inner Source Software Development

InnerSource, also referred to as Inner Source Software Development (ISSD), is the use of open source-like practices and culture within the confines of an organization. The resulting software, however, remains proprietary and for internal use (Capraro et al., 2018; Dorner, 2024; Edison et al., 2020). InnerSource is adopted by organizations of different sizes and sectors, as there is no specific audience, nor is there a limitation, when it comes to sector or number of employees (Dorner, 2024). Since 2002, a steady stream of scientific publications, along with blog and magazine articles, indicates a strong and ongoing interest in this research topic (Capraro & Riehle, 2017). This rise in interest was sparked seeing the numerous benefits that the open source strategy has brought. Naturally adopting such strategy internally could potentially lead to similar results. The InnerSource Commons, founded in 2015, is a great example of the big spike of interest by organizations on the adoption of InnerSource, with over 750 organizations and 3000 Individuals collaborating in this active community. This trend is also evident among major tech companies such as IBM, Google, Microsoft, Hewlett-Packard, and SAP (Wan et al., 2022). The most frequently reported benefits of InnerSource in the literature include reduced development and maintenance costs and efforts, the promotion of software component reuse, knowledge sharing, and better software quality (Capraro & Riehle, 2017; Dorner, 2024; Edison et al., 2020; Wan et al., 2022). Constantino et al. (2023) report other benefits such as enhancement of motivation, engagement, and retention of developers, thus ensuring the longevity of the project. In many instances, the potential advantages of InnerSource are comparable to the rewards organizations have gained from adopting open source (Edison et al., 2020; Wan et al., 2022).

1.2 Transfer Pricing

Having established a clear understanding of InnerSource and its numerous benefits, it is crucial to consider the financial and economic implications of such practices within organizations. This naturally leads to the topic of transfer pricing and the importance of accurate valuation. Transfer pricing is the practice of setting the price for goods, services and intangibles exchanged between different branches of a multinational enterprise (MNE), especially when these branches operate in different countries. This practice is crucial for several reasons including, but not limited to, tax compliance by allocating income and expenses fairly among the entities involved, diminishing the risk of economic double taxation, consistency in tax administration facilitating a fair and equitable tax environment (OECD, 2022). Some companies may even block InnerSource programs due to concerns over tax implications of transferring IP across organizational boundaries (Dillon, 2023). One of the fundamental methods in transfer pricing is the arm's length principle, which requires transactions between related parties to be conducted as if they were between unrelated parties, each acting in their own self-interest. It also ensures that costs are allocated fairly among participants based on their contributions and the benefits they derive (Dorner, 2024). It is important to note that a proper valuation of each entities' contribution plays a vital role in ensuring fair and compliant financial practices when sharing and leveraging software assets, even when it's internally.

1.3 DEMPE Functions

In the context of Transfer Pricing, particularly when dealing with intangible assets such as software, the focus shifts to the Development, Enhancement, Maintenance, Protection and Exploitation (DEMPE) functions. As introduced by the Organisation for Economic Co-operation and Development (OECD) in 2015, These functions are essential for determining how value is created and distributed within a multinational organization, and they serve as the foundation for assessing the appropriate allocation of profits and ensuring compliance with tax regulations (Rudzika, 2018). These functions assist in precisely defining the actual transactions and establishing arm's length pricing for intangibles by clarifying the roles and contributions of various entities within a multinational enterprise (OECD, 2022). Hence, the need for a robust classification system that can categorize InnerSource contributions in alignment with the DEMPE functions. Such a system could facilitate the valuation, as well as compliance with transfer pricing regulations. By classifying contributions according to the DEMPE functions, organizations can better understand the value created by each contribution, ultimately enabling the assignment of a precise dollar amount to these contributions. This approach ensures that internal practices are financially sound, compliant with

international standards, and reflective of the true economic value of the contributions.

1.4 Thesis Structure

In this thesis, the structure is organized as follows: First, the Methodology chapter outlines the approach taken for the literature review, providing a detailed explanation of the research method employed. Following this, the Problem Identification chapter discusses the key challenges addressed by this research and clearly states the objectives of the study. The subsequent chapter, Results, presents the findings of the research, beginning with a detailed exploration of what constitutes an InnerSource contribution and an overview of existing classification systems. This is followed by the presentation of a newly developed classification system, designed to categorize contributions according to the DEMPE functions. The Limitations chapter addresses the constraints encountered during the research. Finally, the thesis concludes with the Conclusion and Outlook chapter, summarizing the key outcomes and proposing potential avenues for future research.

2 Methodology

The methodology to develop a classification system for InnerSource contributions involves a systematic approach. starting with a comprehensive literature search (see Figure 2.1). We refined our search string by piloting it in Scopus and analysing the resulting papers. Once we achieved an accurately refined search string that we believe to cover most of the relevant literature, we moved on to applying it to Scopus and Web of science. Below is the refined search string:

```
TITLE-ABS-KEY ( innersource OR inner*source OR "internal open
source" OR "firm-internal open source" OR "collaborative software
development" OR "internal open collaboration" ) AND TITLE-ABS-KEY
( software ) AND TITLE-ABS-KEY ( contribut** )
```

This search yielded 131 results, we handpicked an additional 5 papers manually. To ensure relevance, we defined specific inclusion and exclusion criteria. We included papers that focus either on InnerSource including all of its different forms, contribution characteristics, classification of contributions. We included papers that are written in English or German, and have been published in the past 10 years. Excluded were papers that focused solely on Open Source without any relevance to our use case, or had a completely different scope.

After an initial screening based on title, abstract and keywords using the defined inclusion and exclusion criteria, 18 papers were selected for a full-text review. Finally, after a thorough reading of the full-text, 16 papers were deemed relevant to our research topic. The relevant literature has been presented in Table 2.1.

However, we recognized that literature alone would not provide a full understanding of the challenges of implementing InnerSource practices. Therefore, we sought to supplement our findings with insights from industry practices. We explored the InnerSource Commons, where companies like PayPal share their experiences and best practices in adopting InnerSource methodologies. Additionally, we examined widely adopted practices that simplify commit message structuring, to see how their principles could be adapted to our classification system.

Combining the relevant literature with the insights from industry practices provided a foundation for understanding the current state of InnerSource research and practices. However, it became apparent through this review that significant gaps exist in the literature, particularly regarding a standardized approach to classifying InnerSource contributions within the DEMPE framework. These gaps highlight the need for a more structured classification system, which will be addressed in the challenges section that follows.

Table 2.1: Relevant Literature

Author	Title	Year
Capraro, Maximilian	The patch-flow method for measuring inner source collaboration	2018
Constantino, Kattiana	Perceptions of open-source software developers on collaborations: An interview and survey study	2023
Dillon, Clare	Who owns this code? Examining Ownership Concepts in InnerSource Practices	2023
dos Santos, Geanderson E.	Commit classification using natural language processing: Experiments over labeled datasets	2020
Hundhausen, Christopher	Combining GitHub, Chat, and Peer Evaluation Data to Assess Individual Contributions to Team Software Development Projects	2023
Lima, Cleyciane	On the Nature of Duplicate Pull Requests: An Empirical Study Using Association Rules	2022
Moreira Soares, Daricélio	What factors influence the lifetime of pull requests?	2021
Rücker de Bassi, Patricia	Measuring Developers' Contribution in Source Code using Quality Metrics	2018
Shen, Chaochao	Git Merge Conflict Resolution Leveraging Strategy Classification and LLM	2023
Wan, Zhiyuan	What motivates software practitioners to contribute to inner source?	2022
Zhao, Guoliang	Improving the pull requests review process using learning-to-rank algorithms	2019

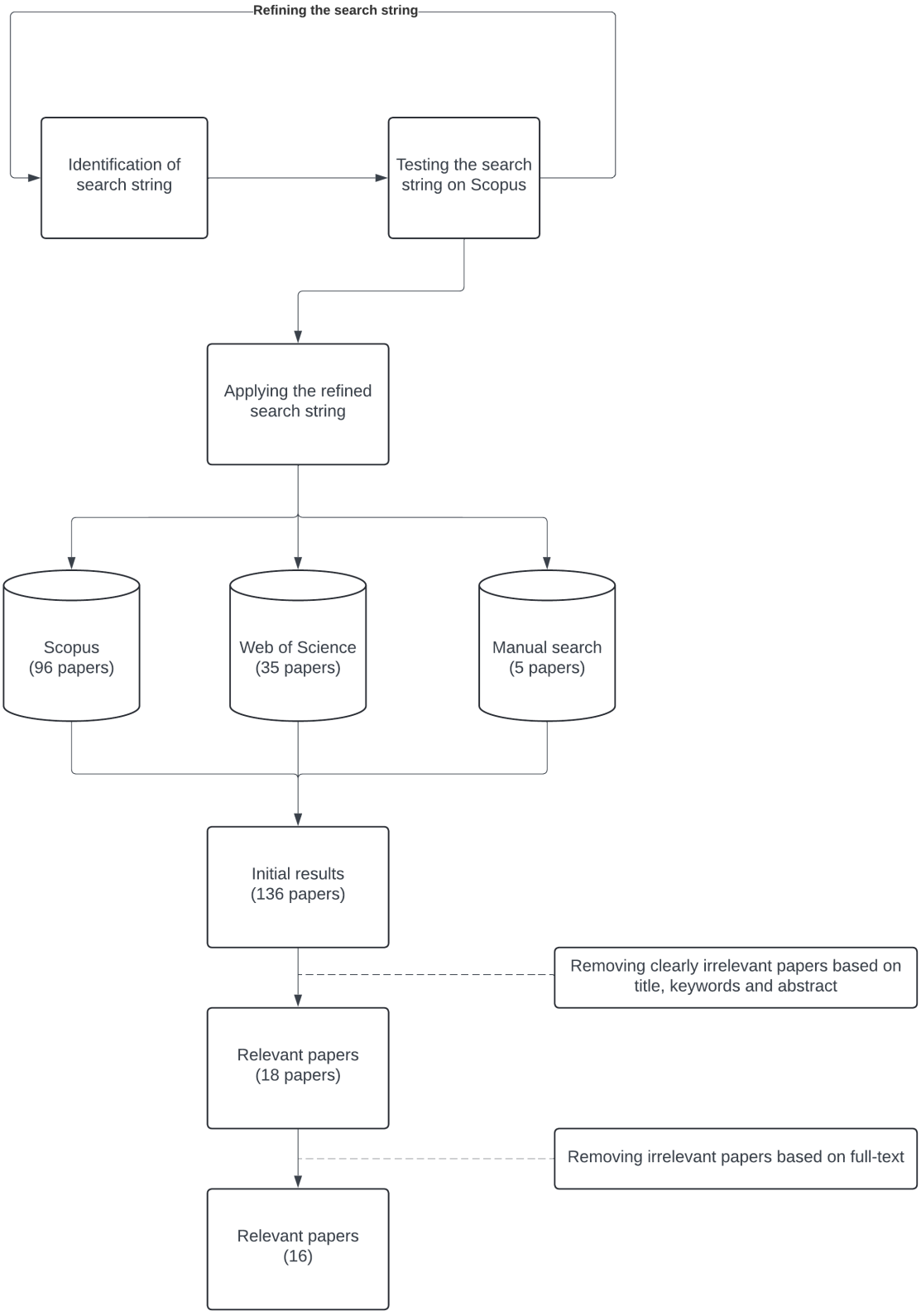

Figure 2.1: Literature Selection Process

3 Problem identification

3.1 Current Challenges

Upon review of the relevant literature, several challenges were encountered that impacted the approach and scope of the study. One of the primary challenges was the fact that the research area, which intersects software engineering and DEMPE, is extremely under-explored. There is a notable scarcity of studies that address this specific combination of topics, making it challenging to find existing work that directly relates to the research question. Given the limited research available on this specific topic, it has been difficult to find quantitative data that could serve as a foundation for developing a classification system. The scarcity of quantitative insights has restricted the ability to create an NLP model based on established research, leading to the need for alternative approaches. Approaches to the topic of software commit classification vary across different studies and practices. This variability has created difficulties in synthesizing a unified approach that could be applied consistently. The diversity of methods and perspectives presents a challenge in achieving consensus on best practices or standards in this area.

3.2 Objective Definition

The primary objective of this thesis is to develop a robust classification system that categorizes InnerSource contributions in alignment with the DEMPE functions. This classification system aims to provide a structured approach to categorizing contributions within the context of transfer pricing, particularly where InnerSource practices intersect with the principles of DEMPE.

The objectives of this thesis are as follows:

- Define InnerSource Contributions.
- Explore Existing Classification Systems.
- Develop a Classification System.

Due to the lack of quantitative data, this thesis will create a specification similar to conventional commit practices. This specification will serve as a guideline for the classification system, ensuring that it is practical and applicable in real-world scenarios.

It is important to note that while this system aims to provide a structured and consistent approach to classifying InnerSource contributions, determining an exact dollar value for each contribution is beyond the scope of this research. Instead, the focus will be on creating a framework that enhances understanding and management of these contributions within the context of transfer pricing, with the help of the DEMPE functions.

4 Research Results

4.1 Findings

4.1.1 InnerSource Contributions

An InnerSource contributor is someone who is not part of the core team responsible for a codebase, but is still more closely aligned with the host team than someone who simply requests a feature (Capraro et al., 2018). These contributors are outsiders who contribute to a codebase without bearing full responsibility for its maintenance (InnerSource Commons, 2024). To contribute to a code-base, be it in InnerSource or Open Source, one needs to review the guidelines set by the host team on contributing. These guidelines are often written on either the README.md file, or written into a standalone CONTRIBUTING.md file. Understanding and following these guidelines is a very crucial step to creating a good contribution. (InnerSource Commons, 2024) Once one has familiarized them-self with the guidelines, they can begin writing commits and creating a pull request. After the code has been reviewed by the host team, it can be integrated into the codebase and featured in the next release. In the InnerSource Commons (2024) Community, a trusted developer that contributes often to a code base can gain the role of Trusted Committer. This role allows the committer to review, approve, and merge contributions from other developers directly into the codebase. Although code contributions are probably the most popular way to contribute to a project. There are many other valuable non-code ways to contribute. For example, you can contribute by testing and reporting bugs, improving documentation, translating content, or even helping with user experience design. As stated before, an InnerSource contribution will typically take form of a pull request, but it could also be a simple commit, a code review or just a quick edit in the documentation. Understanding and classifying these contributions is crucial for effective software project management, as it provides insights into the entire software development life cycle (Dos Santos & Figueiredo, 2020). However, before we can accurately categorize these contributions, it is essential to first identify and understand their defining characteristics.

The smallest form of a code contribution is a commit. It is comprised of several key elements, for example a commit hash, commit message, author information, and timestamp (Chacon & Straub, 2014). Additionally, there is external information related to a commit, such as the code diff and commit history, which may not be stored within the commit itself. Commit messages are frequently analyzed using text analysis techniques to gain insights into the nature of the changes (Dos Santos & Figueiredo, 2020).

Once a contributor is satisfied with the changes they have done, they create a pull request, which is typically composed of multiple commits. The characteristics of a pull request are often a title, description, list of commits, files changed, labels and tags (Hundhausen et al., 2023; Lima & Soares, 2022; Zhao et al., 2019).

After a certain period, a team might decide to push their code into production and publish a new release. This release typically includes key elements such as release notes, a detailed description of the changes, and a designated version number. To ensure consistency and clarity, most teams adhere to a specific standard or specification for versioning their releases.

4.1.2 Classification of Contributions

There is no standard method to classify commits, neither is there a consensus regarding the different types of classification to which a commit refers (Dos Santos & Figueiredo, 2020). There are various methods used to classify commits into pre-defined categories. For example, creating a specification that developers can use to classify and commit their code in a proper way and another approach would be to use Natural Language Processing (NLP).

Conventional Commits is a popular specification that defines a set of rules that create an explicit commit history (Conventional Commits, 2019). Such a specification helps to classify commits into types. For example: fix, feat, breaking change. Other types, based on a shareable commitlint config enforcing Conventional Commits, include chore, docs, style and refactor. Conventional Commits are based on Semantic Versioning 2.0.0 (SemVer), in which "the breaking change" correlates with "Major" in SemVer. SemVer is a powerful specification that is famously used to give meaning to versioning. Each version number takes the format of MAJOR.MINOR.PATCH, in which MAJOR version means that this version has incompatible API changes. Adding new features but keeping backward compatibility would be a MINOR version, and finally a PATCH version would be backward compatible version, in which bug fixes were made. (Semantic Versioning, 2013).

Semantic Commits is a similar specification, that is also based on SemVer and has the same goal as Conventional Commits, classifying commits into different types and creating an explicit commit history. Both Conventional- and Semantic

Commits use a similar set of commit types. The only difference is that Semantic Commits classifies the commits based on the impact they have on the code-base.

Writing release notes in addition to building a code-base can be tedious, if you do not have the ability to automatically generate a list of categorized commits (InnerSource Commons, 2020). Semantic Release relieves developers from this tedious task. It is also based on the SemVer specification and it automates the whole process of releasing, where it determines automatically the next version number, generates the release notes and publishes the package.

4.1.3 Natural Language Processing

Another approach to classify code contributions is using Natural language processing. This approach differs from what we have seen in the previous section, where the aim isn't to enforce a structure that developers should obey, approaches using NLP focus on training a model to classify already existing contributions. Most NLP classification models are based on the commit messages, using text analysis, such as frequency of specific keywords (Dos Santos & Figueiredo, 2020).

Dos Santos and Figueiredo (2020) evaluated five Machine Learning Models that aim to classify commits based on the nature of changes made. Using an ad-hoc literature review, they created a unified dataset by classifying commits into the following categories: corrective, features, perfective, non-functional and unknown. Their proposed model generated by fastText, a library for efficient text classification, has a relatively higher accuracy classifying commits than other similar models.

Characteristics such as the number of commits, files changed and lines changed are a common way to evaluate code contributions, as they are indicative of the nature of the change (Dos Santos & Figueiredo, 2020; Hundhausen et al., 2023; Lima & Soares, 2022; Moreira Soares et al., 2021; Zhao et al., 2019). Thus the use of such characteristics to train the NLP models. Instead of creating a model from scratch, another approach is to use Large Pre-trained Models (LLMs). They are robust systems trained on billions of text documents enabling them to comprehend and generate human-like text outputs (Shen et al., 2023). LLMs are equipped to handle various NLP tasks, including classification of code based on its nature.

4.2 Classification System

4.2.1 Overview of the Classification System

To effectively assess the value of InnerSource contributions, it is essential first to classify these contributions according to the OECD's DEMPE functions. The proposed classification system employs a structured format for commit messages, pull requests, and releases, ensuring that each contribution is clearly categorized under one of the DEMPE functions. The format includes a mandatory prefix that indicates the function, followed by a scope indicating the specific area of the codebase or project affected, and a concise description of the change. This format is applied across all InnerSource contributions, creating consistency and improving traceability throughout the development process.

The general format, should be as follows:

```
<function>(<scope>): <short summary>

[optional body]

[optional footer(s)]
```

Each component of the format is defined as:

- Function: A keyword representing one of the DEMPE functions (e.g., dev, enh, maint, prot, exploit).
- Scope: A noun that describes the area of the codebase affected (e.g., Auth, RootLayout, API).
- Short Summary: A brief description of the contribution (e.g., Add ReactAuthKit).
- Body: An optional section that provides a detailed explanation of the change.
- Footer: An optional section used for metadata related to the contribution (e.g., "Closes #123")

As there is no clear definition of the DEMPE functions in the context of software engineering, we expanded on these functions in Table 4.1 based on their initial description by the OECD.

Table 4.1: DEMPE Functions' Descriptions

Function	Description
Development	This function is used for contributions that introduce new features, create new modules, or significantly refactor existing code to add functionality.
Enhancement	This function is applied to contributions that improve or optimize existing features without introducing entirely new functionality. This includes performance improvements, UI enhancements, and other refinements.
Maintenance	Maintenance encompasses contributions that involve bug fixes, minor refactoring, or technical debt repayment. These changes do not add new features but are essential for the ongoing health of the codebase.
Protection	This function is reserved for contributions that focus on security, legal protections, or intellectual property management. This includes adding security features, implementing encryption, or making changes related to licensing.
Exploitation	Exploitation refers to contributions that relate to the monetization, deployment, or release of the software. This includes packaging, marketing, and other activities aimed at capitalizing on the software.

4.2.2 Examples

Here are a few examples of how contributions can be properly categorized and classified according to the classification system:

Development commit with commit message only

```
dev(Auth): Add OAuth2 Authentification
```

Enhancement commit with multi-paragraph body and footer

```
enh(UI): Improve dashboard responsiveness

- Updated the CSS grid layout to ensure responsiveness on
    mobile devices.
- Optimized loading times for dashboard components.

#References Issue #76
```

Maintenance commit with a brief body

```
maint(Logging): Fix typo in log output

- Corrected the spelling mistake in the error log messages to
improve clarity.
```

Protection commit with footer

```
prot(Database): Add input validation against SQL Injections

#Closes Security Issue #12
```

Exploitation commit with a detailed body and multiple footers

```
exploit(Deployment): Prepare release 3.2.1 for production

- Updated the deployment scripts to support the new
infrastructure.
- Verified that all critical services are included in the
deployment package.
- Conducted a final round of tests to ensure stability
post-deployment.

#Release Version 3.2.1
#Related to Deployment Task #145
```

4.2.3 Relevance to Semantic Versioning

As stated in the findings, the SemVer specification differentiates between three different types of versions; MAJOR, MINOR and PATCH. We can map these version increments according to the DEMPE functions, resulting in Figure 4.1.

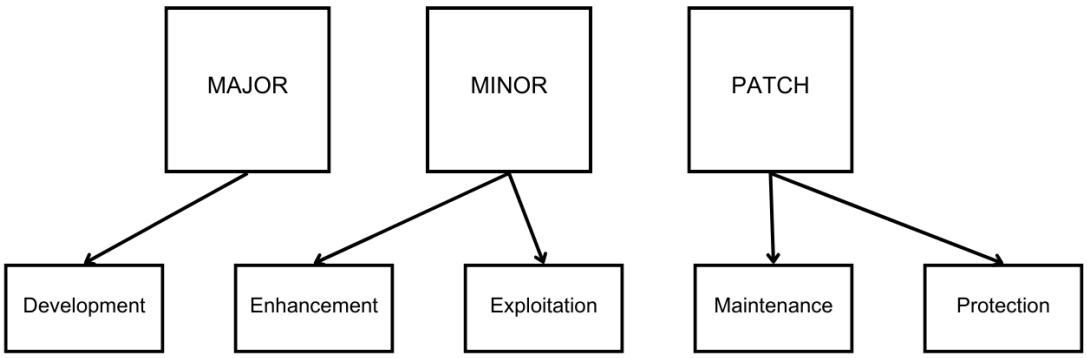

Figure 4.1: SemVer and the DEMPE functions

The reasoning behind Figure 4.1 is as follows:

- `MAJOR` Version Increment:
  - Development: Introducing significant new features or making major refactoring changes is a MAJOR version increment. Unless it is backward-compatible.
- `MINOR` Version Increment:
  - Enhancement: Enhancements that add new functionality or improve existing features in a backward-compatible manner would typically result in a MINOR version increment.
  - Exploitation: Even though exploitation activities like deployment preparations do not typically change the software itself, introducing new releases that include new features or enhancements would likely be a MINOR version increment.
- `Patch` Version Increment:
  - Maintenance: Maintenance activities that involve bug fixes or minor adjustments to ensure the software functions as expected without adding new features or altering existing ones should result in a PATCH version increment.
  - Protection: Security fixes or other protection-related updates that do not change the software's existing behavior but improve security or compliance would typically lead to a PATCH version increment.

4.2.4 Application to Different Contribution Types

The classification system should be applied consistently across different types of InnerSource contributions. Starting with a commit, the commit message should begin with the relevant DEMPE function keyword, ensuring that even individual code changes are clearly categorized. This structure is enforced through Git hooks that validate commit messages against the defined format. When creating a pull request, the title and description should follow the same format. This ensures that the purpose of the pull request is clear to reviewers and aligns with the overall project goals. Lastly, the release notes are structured using the DEMPE functions to categorize the changes included in each release. This makes it easier for stakeholders to understand the nature of the updates and the areas of the project that have been impacted.

Let's not forget non-code contributions. As stated in the findings section, non-code contributions also play a vital role in maintaining a codebase. These contributions, which may include documentation, design assets, process improvements, and community management activities, should also be classified according to the DEMPE framework. By applying the same structured approach to non-code contributions, the system recognizes and values the full spectrum of work that supports the development and sustainability of the project. This approach ensures that all forms of contributions are acknowledged.

4.2.5 Tooling and Automation

To facilitate and automate the use of the classification system, there are a few key tools and automation features that could be developed and used.

Commit Message Linter
One of the first steps would be to introduce a Commit Message Linter. As demonstrated by its successful use in the development of Angular, this tool functions as a pre-check for commits, ensuring that each commit message adheres to the structure set out by the classification system. This approach helps maintain consistency and proper classification from the outset, reducing the likelihood of errors and omissions in the commit history (Angular Team, 2024).

CI/CD pipeline
Another important aspect is integrating the classification system into the CI/CD pipeline. Ensuring that the pipeline can automatically identify any contribution that doesn't adhere to the standards of our classification system. If a contribution doesn't meet the required format, the build would fail, prompting the contributor to correct it.

Git Templates
Providing Git Templates to the developers is another way to facilitate the use

and adoption of the classification system. The templates should provide a pre-structured format for the commit messages, guiding the developers in applying the classification system correctly. This should reduce errors and make sure contributions are consistent across board.

Automated Changelog Generation
Finally, Automated Changelog Generation would be a crucial tool for organizing and summarizing changes in each release. This tool should automatically categorize changes according to the DEMPE functions, making it easier to create clear and concise release notes. This would help users quickly understand the updates and their impact on the project.

4.2.6 Validation

Validating the classification system for InnerSource contributions is a crucial step to ensure that it effectively meets the needs of the adopting organizations and provides the intended benefits. This validation process should help confirm that the system is effective and accurate in categorizing the contributions.

To begin, the system should be validated through Pilot Testing. Pilot testing is widely recognized as a preliminary phase in system validation that allows for the identification of potential issues and improvements before broader implementation (Teijlingen & Hundley, 2002). Introducing the system to a small team working on a specific project will allow for an assessment of its performance on a smaller scale. Observing how ease to adopt it is and how the team performs in a controlled environment is a great way to begin the validation. This pilot phase will also allow the gathering of valuable feedback from the team. Based on that feedback, we can make adjustments to refine the system before testing it on a bigger scale. One should keep in mind that tracking Performance Metrics is vital to a successful adoption of the system. Monitoring specific indicators such as the adoption rate, error rate and overall efficiency can be a quantifiable way to measure the success of the system as well as its impact.

Overall, the validation process for the classification system involves a combination of pilot testing and performance metrics tracking to ensure that the system is ready to launch on a bigger scale. It is important also to make sure that system serves the intended goal and is easy to adopt by different unique teams.

5 Limitations

While this thesis offers a significant contribution by providing a classification system for InnerSource contributions according to the DEMPE functions, it is important to recognize the limitations that may affect the generalizability and the applicability of the findings. A key limitation of this thesis is the lack of available literature on DEMPE functions, especially in the context of software development and InnerSource practices. We have used Scopus to make a comparison of literature availability of similar topics (see Table 5.1).

Table 5.1: Comparison of Literature Availability

Topic	**Number of Relevant Papers**
DEMPE Functions in Software	0
OECD's DEMPE Functions	11
IP Valuation in Software	196+
Taxation and Accounting Software Practices	11.388+

This shortage of research may have limited the theoretical basis for the classification system and required the use of broader interpretations of DEMPE functions. The implementation of InnerSource practices varies widely across organizations, depending on their size, industry, and technological infrastructure. This variability could limit the generalizability of the classification system, as it may perform differently in organizations with different InnerSource maturity levels or technological environments. A significant limitation of the proposed classification system is that it is primarily designed for prospective application. It is intended to classify future InnerSource contributions based on their alignment with DEMPE functions, rather than to retroactively classify commits that have already been made. This orientation toward future contributions means that the system may not be suitable for organizations seeking to evaluate or reclassify historical contributions. As a result, its utility is constrained to forward-looking scenarios, where planning and categorization of contributions can be aligned with DEMPE functions from the outset. This research was conducted within the scope of a bachelor thesis, which inherently imposes certain constraints, particularly regarding time

and resources. The limited time frame available may have restricted the depth of analysis and the range of literature collection.

6 Conclusion and Outlook

This thesis defines InnerSource contributions and explores the different ways to classify them. Which gave us the foundation to develop our own classification system based on SemVer. By mapping contributions to the OECD's DEMPE functions, ensuring that all types of contributions were systematically documented.

Implications
The adoption and successful implementation of this classification system has several important implications for organizations using InnerSource practices. By providing a clear framework for categorizing contributions, this system not only ensures better documentation practices but also plays a critical role in enhancing tax compliance and financial management. Proper categorization is essential for accurately determining the value of contributions, which is crucial for maintaining compliance with transfer pricing regulations and other tax-related requirements.

Beyond its critical role in tax compliance, the system enhances transparency and accountability in financial management. A standardized approach to categorization allows finance and tax teams to track the financial impact of contributions more effectively, ensuring that they are valued appropriately and in line with regulatory expectations.

In summary, the implementation of this classification system strengthens the organization's ability to manage complex global operations, ensuring that contributions are valued accurately and that the organization remains compliant with international tax and financial standards.

Future Work
There are several areas for future work that can enhance the classification system. Enhancements in tooling and automation, such as the development of dynamic Git templates, commit message linters, and CI/CD integration, could further streamline the use of the system.

To ensure the system's effectiveness, validation through pilot testing and performance metrics is a must. Testing the system in a real-world setting to gather feedback and assess its impact, is needed. It's also necessary to setup a continu-

ous feedback loop with iterative improvement to ensure the system meets the needs and provides the benefits sought after by the organisations.

Finally, after validating this system though comprehensive testing, It could play a crucial role in accurately valuing InnerSource contributions. The system can simplify and streamline the process of assessing the economical value of InnerSource contributions. This, in turn, could facilitate more transparent and efficient taxation, ensuring that contributions are appropriately recognized and accounted for in financial and legal contexts.

References

Angular Team. (2024). Angular commit message guidelines [Accessed: 2024-09-01]. https://github.com/angular/angular/blob/main/CONTRIBUTING.md#commit

Capraro, M., Dorner, M., & Riehle, D. (2018). The patch-flow method for measuring inner source collaboration. In A. Zaidman, Y. Kamei & E. Hill (Eds.), *Proceedings of the 15th international conference on mining software repositories* (pp. 515–525). ACM. https://doi.org/10.1145/3196398.3196417

Capraro, M., & Riehle, D. (2017). Inner source definition, benefits, and challenges. *ACM Computing Surveys*, *49*(4), 1–36.

Chacon, S., & Straub, B. (2014). *Pro git* (2nd). Apress. https://git-scm.com/book/en/v2

Constantino, K., Souza, M., Zhou, S., Figueiredo, E., & Kästner, C. (2023). Perceptions of open–source software developers on collaborations: An interview and survey study. *Journal of Software: Evolution and Process*, *35*(5). https://doi.org/10.1002/smr.2393

Conventional Commits. (2019). Conventional commits v1.0.0 [Accessed: 2024-09-01].

Dillon, C. (2023). Who owns this code? examining ownership concepts in innersource practices. *2023 IEEE/ACM 1st International Workshop on InnerSource Software Development (InnerSoft)*, 2–3. https://doi.org/10.1109/InnerSoft59330.2023.00009

Dorner, M. (2024). Sustaining arm's length cost allocations for highly integrated development functions: An explorative case study of transfer pricing for innersource communities.

Dos Santos, G. E., & Figueiredo, E. (2020). Commit classification using natural language processing: Experiments over labeled datasets. *Conferencia Iberoamericana de Software Engineering*, 110–123.

Edison, H., Carroll, N., Morgan, L., & Conboy, K. (2020). Inner source software development: Current thinking and an agenda for future research. *Journal of Systems and Software*, *163*, 110520.

Hundhausen, C., Conrad, P., Adesope, O., & Tariq, A. (2023). Combining github, chat, and peer evaluation data to assess individual contributions to

team software development projects. *ACM Transactions on Computing Education*, *23*(3), 1–23. https://doi.org/10.1145/3593592

InnerSource Commons. (2020). *Innersource patterns* [Accessed: 2024-09-01]. https://innersourcecommons.org/patterns

InnerSource Commons. (2024). Innersource commons [Accessed: 2024-09-01]. https://innersourcecommons.org/

Lima, C., & Soares, D. (2022). On the nature of duplicate pull requests: An empirical study using association rules. In M. Maia, F. Dorça, R. Araújo, T. E. Colanzi & E. D. Canedo (Eds.), *Proceedings of the 16th brazilian symposium on software components, architectures, and reuse* (pp. 68–75). ACM. https://doi.org/10.1145/3559712.3559722

Moreira Soares, D., de Lima Júnior, M. L., Murta, L., & Plastino, A. (2021). What factors influence the lifetime of pull requests? *Software: Practice and Experience*, *51*(6), 1173–1193. https://doi.org/10.1002/spe.2946

OECD. (2022). *Oecd transfer pricing guidelines*. https://doi.org/10.1787/0e655865-en

Rudzika, K. (2018). *Dempe explained*. https://www.royaltyrange.com/home/blog/dempe-explained

Semantic Versioning. (2013). Semantic versioning 2.0.0 [Accessed: 2024-09-01].

Shen, C., Yang, W., Pan, M., & Zhou, Y. (2023). Git merge conflict resolution leveraging strategy classification and llm. *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, 228–239. https://doi.org/10.1109/QRS60937.2023.00031

Teijlingen, E. R. V., & Hundley, V. (2002). The importance of pilot studies. *Social Research Update*, (35). http://sru.soc.surrey.ac.uk/SRU35.html

Wan, Z., Xia, X., Zhang, Y., Lo, D., Zhou, D., Chen, Q., & Hassan, A. E. (2022). What motivates software practitioners to contribute to inner source? In A. Roychoudhury, C. Cadar & M. Kim (Eds.), *Proceedings of the 30th acm joint european software engineering conference and symposium on the foundations of software engineering* (pp. 132–144). ACM. https://doi.org/10.1145/3540250.3549148

Zhao, G., Da Costa, D. A., & Zou, Y. (2019). Improving the pull requests review process using learning-to-rank algorithms. *Empirical Software Engineering*, *24*(4), 2140–2170. https://doi.org/10.1007/s10664-019-09696-8