Noname manuscript No.
(will be inserted by the editor)

Describing Globally Distributed Software Architectures for Tax Compliance

Michael Dorner · Oliver Treidler · Tom-Eric Kunz · Ehsan Zabardast · Daniel Mendez · Darja Šmite · Maximilian Capraro · Krzysztof Wnuk

the date of receipt and acceptance should be inserted later

Abstract

Background: The company-internal reuse of software components owned by organizational units in different countries constitutes an implicit licensing across borders, which is taxable. This makes tax authorities a less known stakeholder in software architectures.

Objective: Therefore, we investigate how software companies can describe the implicit license structure of their globally distributed software architectures to tax authorities.

Method: We develop a viewpoint that frames the concerns of tax authorities, use this viewpoint to construct a view of a large-scale microservice architecture of a multinational enterprise, and evaluate the resulting software architecture description with a panel of four tax experts.

Results: The panel found our proposed architectural viewpoint properly and sufficiently frames the concerns of taxation stakeholders. However, unclear jurisdictions of owners and potentially insufficient definitions of code ownership and software component introduce significant noise to the view that limits the usefulness and explanatory power of our software architecture description.

Conclusion: While our software architecture description provides a solid foundation, we believe it only represents the tip of the iceberg. Future research is necessary to pave the way for advancements in tax compliance within software engineering.

Keywords software architecture, taxation, globally distributed software development, microservices, tax compliance

Michael Dorner, Ehsan Zabardast, Daniel Mendez, Darja Šmite, and Krzysztof Wnuk
Blekinge Institute of Technology
E-mail: {michael.dorner | ehsan.zabardast | daniel.mendez | darja.smite | krzysztof.wnuk}@bth.se

Oliver Treidler, Tom-Eric Kunz, and Maximilian Capraro
Kolabri
E-mail: {oliver | tom | max}@kolabri.io

Daniel Mendez
fortiss
E-mail: mendez@fortiss.org

1 Introduction

Code is not tied to a physical, geographical location due to its intangible nature. The code owner of a software component, however, is. Code ownership refers to assigning responsibility and accountability for a particular piece of code or software component to an individual, a team, or an organizational unit within an organization. However, the geographical location of those software component owners has a legal implication that is too often underestimated: Using software components with owners in different countries is taxable as it constitutes from a taxation perspective an—often implicit—form of licensing (Dorner et al. 2024; OECD 2022). To comply with the international standard of taxation (OECD 2022), multinational enterprises must be able to report this implicit license structure of their software architectures from a geographical perspective. However, there is no architectural viewpoint that frames these concerns of tax authorities as stakeholders towards a globally distributed software architecture.

In this study, we aim to understand the concerns of tax authorities as stakeholders towards software architectures in multinational enterprises and seek an answer to the research question:

How can multinational enterprises describe the implicit license structure of their globally distributed software architectures to tax authorities?

Therefore, we developed a viewpoint that frames the concerns of tax authorities, use this viewpoint to create a view of a large-scale microservice software architecture of a multinational enterprise, and evaluate the resulting software architecture description with a panel of tax experts.

The main contribution of this study is the presentation and evaluation of a novel software architecture description for tax compliance.

Throughout this paper, we use the terminology and definitions of ISO/IEC/IEEE 42010:2022 . In particular, we distinguish between architecture description, architecture view, and architecture viewpoint: An *architecture description* is the work product used to express software architectures and is a tangible representation of information provided to the stakeholders. An *architecture view* (or simply, *view*) is an information part comprising a portion of an architecture description. An architecture view expresses the architecture of the system of interest in accordance with an *architecture viewpoint* (or simply, *viewpoint*). A viewpoint is a set of conventions for the creation, interpretation, and use of a view. The software architecture description under evaluation has only one view.

The remainder of the article is structured as follows: After establishing the background and related work in Section 2 and discussing our study design as a judgment study in Section 3, we define the viewpoint (Section 4.2) and how we use it to generate a view for the concern of tax authorities (Section 4.3). In Section 5, we present the evaluation results for our novel viewpoint (Section 5.1, the view (Section 5.2), and the overall software architecture description (Section 5.3) in detail. After critically reflecting on the evaluation results in Section 6 and the limitations of our study in Section 7, we conclude the paper with a summary of our findings and the implications for research and practice before outlining future work in Section 8.

At this point, we would like to emphasize that we focus on a software-engineering perspective on tax compliance. An in-depth legal and economic discussion and evaluation of our work is beyond our reach.

2 Related Work

Since this article connects taxation with software architecture distributed across multiple countries, we provide related work on globally distributed software development and the emerging field of taxation in software engineering in the following subsections.

2.1 Globally Distributed Software Development

Globally distributed software development, especially in large enterprises, is a common practice. Tough competition and a lack of resources motivated many software companies to look for ways to optimize their costs by establishing their own subsidiaries in lower-cost premises locally and nationally or even in other countries or by teaming up with allies that can help in delivering products more cost-effectively. As a result, distributed working across national boundaries has become widespread (Ruhe and Wohlin 2014). However, cross-border collaborations are far from being challenge-free. They are recognized to be considerably more complex than the most complicated one-site development endeavors due to the significant communication and coordination overhead. The solutions to the problems of global projects have evolved through the years. They could be characterized by a long journey from initial problematic experiences of collaborative work through strictly independent work in distributed sites to the evolved ways of working across borders.

Architectural modularization was one of the most prominent early solutions to the coordination and communication challenges in global software engineering. In 1968, Conway published his observation that software architecture cannot be separated from the structure of the organization that develops the software, known as Conway's law (Conway 1968). Inspired by this observation, architectural modularization has emerged as a way to minimize dependencies, structure and allocate tasks, and encapsulate the work in a single geographic development site as much as possible so that communication, coordination, and synchronization needs are minimized (Cataldo et al. 2008; Herbsleb and Mockus 2003; Herbsleb and Grinter 1999a). Although the modularity approach to coordination has proven very useful, it has many limitations (Cataldo and Herbsleb 2008; Šmite 2014; Clerc et al. 2007). While some researchers advocate that strict modularization of work is the possibly best way to alleviate the challenges of distributed work, others argue that work in isolation has many disadvantages (Cataldo and Herbsleb 2008), such as redundant work, cheap and dirty architectural decisions, misplaced functionality (Tureček et al. 2010; Šmite 2014) and integration problems (Kwan et al. 2011; Herbsleb and Grinter 1999b). Software components are never truly independent and even the best designs are never error-free, while changes are never completely predictable (Herbsleb and Grinter 1999b).

Further, strict code ownership is also found to introduce communication overhead and delays (Clerc et al. 2007). For these reasons, and due to the increased use of agile methods, there has been an increasing interest in collaboratively developing software products. This includes a shift from modularization based on geographically distributed locations towards componentization through microservices (Newman 2021) on a team level, as well as a trend towards establishing collective ownership of source code and other documents to allow everyone in the project to work on any model or artifact (Clerc et al. 2007). Collaborative work includes keeping track of which team owns and does what (e.g., through Scrum of Scrum fora or organizing cross-unit demos), submitting contributions to other's components and reviewing each other's code (Dorner et al. 2024; Šmite et al. 2023), and fostering reuse of artifacts to reduce redundant work.

2.2 Taxation in Software Engineering

Taxation in software industry has been debated for many decades (OECD 2015). The problem with taxing the final result of software engineering, the software product or service, for example, has shown to be challenging to tackle and is still subject to ongoing and broad discussion (Olbert and Spengel 2017). This debate has also reached software engineering, the construction of the software product; at first in the context of inner source, use of open source software development practices and the establishment of open source-like communities within an organization (Capraro 2020). To address the taxation challenges in inner source software development, Buchner and Riehle proposed applying a so-called cost-plus approach (Buchner and Riehle 2022). However, Treidler et al. showed that such a cost-plus approach is not applicable since inner source does not reflect the function and risk profile, the starting point for any transfer pricing discussion, for a cost-plus approach (Treidler et al. 2024a).

Dorner et al. identified the use of software components owned by teams or individuals who represent separate geographically distributed subsidiaries of the same enterprise as a type of licensing—a key intangible in software engineering that is taxable (Dorner et al. 2024). We will explain the concerns of tax authorities on software architecture in Section 4.1, in laymen's terms and with a simplified example of a multinational enterprise developing software, how tax authorities become a stakeholder in a globally distributed software architecture.

The very recent litigation from 2023 between Microsoft and the US tax authorities (IRS), in which the IRS alleges that Microsoft owes an additional $28.9 billion in tax from 2004 to 2013, plus penalties and interest, demonstrates the importance of tax compliance for multinational enterprises developing software. For the interested reader, we recommend our analysis of the Microsoft case from a legal perspective (Treidler et al. 2024b).

3 Study Design

Our study consists of two parts as Figure 1 highlights:

1. An *exploratory* part, in which we develop a viewpoint to enable software-developing multinational enterprises to report their implicit license structure derived from the author team's expertise and understanding of the topic and construct a view of a multinational enterprise's software architecture
2. An *confirmatory* part, in which we evaluate the resulting software architecture description for a large-scale software architecture of a multinational enterprise with a panel of tax experts.

As the main contribution of this study, we focus on the evaluation (confirmatory) part of our research on tax compliance in the context of software architecture. To evaluate the software architecture description for its intended purpose, we conducted semi-structured interviews with four tax experts which makes our study a judgment study. In a judgment study, researchers gather empirical data from a group of participants who are asked to judge or rate behaviors, to respond to a request or "stimulus" offered by a researcher, or to discuss a given topic of interest (Stol and Fitzgerald 2018). Stol and Fitzgerald (2018) compare judgment studies to a courtroom: a panel of participants (the jury) is carefully and systematically selected. In a courtroom, evidence is presented (a stimulus) and eventually the jury returns a verdict. The setting itself (i.e., the courtroom) is only manipulated to the extent that it aims to be neutral and not distract the participants from the matter at hand (i.e.,

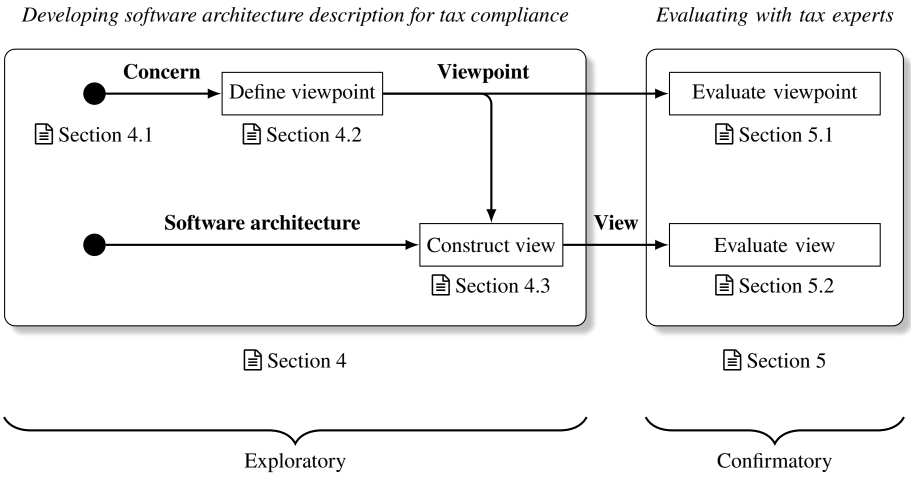

Fig. 1 An overview of the structure of our study: (1) creating a software architecture description with a viewpoint derived from the concerns of tax authorities and a view of an empirical software architecture using the viewpoint (exploratory) and (2) evaluating the software architecture description with a panel of tax experts (confirmatory).

Table 1 An overview of the interviewees and their backgrounds.

ID	Role	Background
A	Tax auditor	Former tax auditor, majoring in international taxation, OECD advisor for transfer pricing from 2016 to 2018, currently not involved in transfer pricing
K	Tax advisor	Transfer pricing advisor, previously transfer pricing expert at one of the Big Four accounting firms for 20 years
M	Tax auditor	Tax auditor and key expert on transfer pricing for other tax auditors for more than eight years
R	Tax advisor	Managing director at a transfer price consulting firm; tax inspector before, 35 years of experience in international taxation

the case). The goal of a judgment study is to seek generalizability over the responses, rather than generalizability to a population of actors (Stol and Fitzgerald 2018).

Therefore, we systematically sampled our experts from both opposing perspectives in a tax audit:

– the company perspective as the provider represented by a tax advisor
– the tax authority perspective as the recipient of such documentation represented by a tax auditor

We selected the participants from our professional network. None of the participants were compensated for their participation. Additionally, none of the participants are currently working or have previously worked with any members of the author team, neither in an academic nor in a professional context. Table 1 provides an overview of our interviewees and their backgrounds.

The stimulus for the panel of four tax experts is a software architecture description of the implicit license structure of a globally distributed software architecture of a multinational enterprise.

We ensured privacy and confidentiality for all actors (i.e., the involved companies, tax authorities, and tax advisors) by conducting the interviews separately and removing all company-specific information. We also altered the geographical distribution of the subsidiaries to make our company not identifiable in any sense on the one hand but also to give the reader a better impression of globally distributed software architecture on the other hand. The interview guidelines are publicly available[1].

We conducted two pilot interviews with the author team and a third tax auditor (who was not part of the panel) to address didactical challenges since the panel has no software engineering but a taxation background. A pair of a co-author with a taxation background and a co-author with a software engineering background conducted the interviews with the tax experts virtually between 2023-10-24 and 2023-11-03. We recorded all interview sessions. The interviews were held in English and German.

As our interview guidelines show, large parts of the interviews were dedicated to providing contextual software engineering understanding for the tax experts. Therefore, we translated and paraphrased only the answers from the interviewees. We analyzed the interviews in a team of two: While one researcher of the interview team analyzed the recorded interviews using thematic analysis (Clarke and Braun 2017), the second one double-checked findings and mappings in a separate session. To avoid translation mistakes and misunderstandings, the final version of the paper was sent to each interviewee, asking for confirmation.

4 A Software Architecture Description for Tax Compliance

In the following subsections, we first provide a gentle introduction to international taxation for software engineers in Section 4.1. We then derive based on our interdisciplinary experiences and opinions the viewpoint, which is the set of conventions for creating, interpreting, and using an architecture view to frame the concern of tax authorities (Section 4.2). In Section 4.3, we apply then the viewpoint to a globally distributed software architecture of a multinational enterprise to be able to evaluate a full software architecture description with the panel of tax experts in Section 5.

4.1 International Taxation Standards as Concern and Tax Authorities as Stakeholder

Consider the fictional *devnullsoft Group*, a multinational enterprise that develops and sells a software product. The software is split up into 18 components. Each component has an owning team that is responsible and accountable for its components.

The teams are assigned to one of the legal entities of the enterprise in three countries depending on the location of the team members: *devnullsoft GmbH* in Germany, *devnullsoft AB* in Sweden, and *devnullsoft Ltd.* in the UK. In Figure 2, we provide a schematic overview of a simplified globally distributed software architecture with owning teams assigned to three different subsidiaries in Germany, Sweden, and the UK. Using code components across different jurisdictions is taxable.

Only the German *devnullsoft GmbH* sells this resulting product to customers. Without any further consideration, solely the German *devnullsoft GmbH* generates profits, which are then fully taxed in Germany according to German law. The Swedish and English tax authorities are left out in the cold because *devnullsoft AB* and *devnullsoft Ltd.* have no share

[1] `https://zenodo.org/records/10214942`

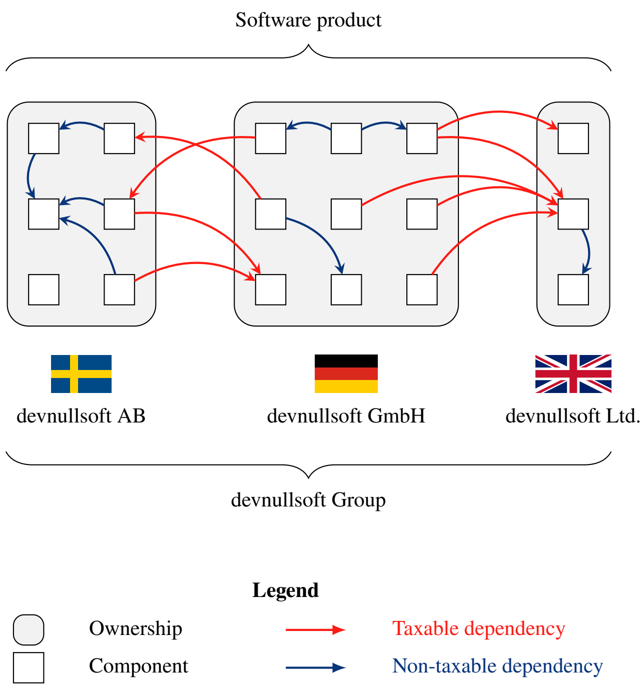

Fig. 2 A schematic overview of a globally distributed software architecture developed by the fictional *devnullsoft Group*, a multinational software enterprise with three subsidiaries developing software. Using code components across different jurisdictions is a form of licensing and, thus, taxable.

of the profit that could be taxed in Sweden or the UK, although both subsidiaries maintain, test, and develop, or simply own components that are crucial for the software product that made the success of the software possible.

To avoid this scenario and to provide a common ground for international taxation, reducing uncertainty for multinational enterprises, and preventing tax avoidance through profit shifting, nearly all countries in the world agreed on and implemented the so-called *arm's length principle* as defined in the *OECD Transfer Pricing Guidelines for Multinational Enterprises and Tax Administrations* (OECD 2022). This has become the guiding principle and the de-facto standard for the taxation of multinational enterprises that requires associated enterprises to operate as if not associated and regular participants in the market from a taxation perspective. To comply with the arm's length principle, multinational enterprises must pay a so-called *transfer price*. Transfer prices are the prices at which an enterprise transfers physical goods or—for software much more relevant—intangibles or provides services to associated enterprises. Those transfer prices are established on a market value basis and aim to avoid profit shifts from high to low tax regions. Although other types of intangibles, like cross-border code contributions, code reviews, bug reports, etc., are also relevant from a taxation point of view, we focus in this article on another highly relevant type of intangible in software engineering: licenses.

Although maybe not even explicit, the company-internal use and reuse of software components is, in fact, an instance of licensing. A software license is a legal instrument governing the use or redistribution of software. Software licenses and their obligations are widely discussed in the context of open-source software. But in contrast to open source, components and their use are protected by the confines of a company. Third parties are excluded from any use or redistribution right of the software components. Without further considerations—i.e., transfer prices—this exclusion contradicts the arm's length principle and makes multinational enterprises like our *devnullsoft Group* not compliant.

In this way, tax authorities become stakeholders in globally distributed software architectures as they have the concern of identifying cross-border reuse of software components as an instance of taxable licensing. Determining the actual transfer pricing for the cross-border reuse of software components is a vast and separate challenge on its own and not part of this work. Our work focuses on the architectural description comprising all necessary information for reporting the software architecture to tax authorities.

To conclude, the concern of tax authorities on software architecture is the (potentially implicit) license structure of the software-intensive product within the multinational enterprise. From this concern, we can derive three main questions:

1. How is the software-intensive product structured?
2. What legal entities (implicitly) license those substructures of the software?
3. Where are those legal entities geographically located?

All three questions must be answered by a software architecture description to ensure tax compliance for multinational enterprises.

4.2 Architecture Viewpoint

To address the concern of tax authorities and to answer the three derived questions, our viewpoint requires four elements: Defining the component and component owner, decomposing the component structure of the software product, identifying component owners, and identifying the jurisdictions of the component owners.

4.2.1 Defining component and component owner

A software component is a self-contained, reusable piece of software that encapsulates the internal construction and exposes its functionality through a well-defined interface so other components can use the functionality. That enables software components to be used, developed, tested, and maintained individually by different teams and separately from the rest of the software product. Software components can take many forms, including libraries, modules, classes, functions, or even entire applications.

Component ownership refers to the concept of assigning responsibility and accountability for a particular software component to an individual or an organizational unit within an organization. Although there are different notions of code ownership (for example, strong, weak, or collective code ownership (Fowler 2006)), component ownership is more strictly defined in this context: We require only one organizational unit to be ultimately responsible and accountable for a software component.

The definition of a component or a component owner is company-specific and may vary. Thus, an in-depth rationale for the definitions in the context of the reporting company is required.

4.2.2 Decomposing the component structure of the software product

The software product must be decomposed into the component structure to reveal the component structure and the dependencies of software components. This results in a dependency graph reflecting the dependencies between the software components. The type of dependency depends on the definition of a software component. Therefore, a rationale is required.

4.2.3 Identifying component owners

Each component must be assigned to one owner. This usually results in a table of components and their owners. As discussed previously, the mapping of owner to component is a one-to-many mapping.

4.2.4 Identifying jurisdictions of component owners

Each organizational unit (or individual) owning a software component must be assigned to one jurisdiction. In most cases, a jurisdiction is the country where the local subsidiary is located. This usually results in a table of component owners and countries. As discussed previously, mapping an owner to a country is one-to-one.

4.3 Architecture View

In this section, we discuss the practical view construction using the presented viewpoint at a large multinational enterprise. Understandably, we are not able to describe the case and the data collection in detail to maintain the anonymity of our case company, which was used as a source for creating our view. However, we may well argue that our case company is exemplary for a multinational enterprise developing software. In brief, to create the view, we extracted the software architecture and the team information from an internal system that extracts the component structure of the system from the code repositories on a daily basis.

We use microservices as the most suitable abstraction layer for software components for the specific software architecture in our case. A microservice represents a discrete piece of a larger application's functionality and operates as a self-contained unit. Microservices are typically designed to be lightweight, modular, and independently scalable, allowing for greater flexibility and agility in software development (Newman 2021). Therefore, microservices meet the definition of a software component.

The software architecture of the software product consists of 2560 different microservices. We consider only microservices in production; experimental or other non-production microservices are excluded. Microservices in production imply that those microservices face the customer directly or indirectly.

In our case company, a component owner is responsible and accountable for its software component which includes functional and non-functional requirements, such as security, reliability, and quality, and extends throughout the software lifecycle, including development, testing, and maintenance. It also involves all aspects of operating that code in production, such as deployment, monitoring, troubleshooting, and remediation of operational and security incidents.

Table 2 An equivalent tabular representation of Figure 3 but including unknown dependencies.

		Component owner					
		DEU	GBR	NLD	FRA	USA	N/A
Component user	DEU	2	2	0	0	0	4
	GBR	15	164	2	261	43	141
	NLD	3	6	19	11	5	8
	FRA	24	108	21	4069	850	1767
	USA	14	24	15	1130	1648	642
	N/A	27	70	14	2283	970	2171

In our case company, each microservice is usually owned by a single team. We excluded 42 microservices (1.64%) each owned by a single person, because the locations of individuals were not accessible for this research due to privacy concerns. Those remaining 2518 microservices have 16533 dependencies between them and are owned by 336 different teams.

The geographical locations of the teams are self-assigned: Teams report their location in a central system. We find this self-reporting most realistic in the world of distributed and remote teams. However, we were not able to retrieve or confirm the actual geographical location of 140 teams (41.16%) with this self-reporting approach. This is mostly due to remote and distributed teams that do not have a single location that we could assign to a single jurisdiction. This affected 8097 use relationships, i.e., 48.94% of all use relationships.

Due to the large amount of data, we present the architecture view in two aggregated visualizations, as a graph (Figure 3) and as a table Table 2. The directed graph depicts jurisdictions (countries) as vertices and dependencies as directed edges. An edge goes from the component user to the component owner. We aggregate the number of dependencies between countries as edge weights. This graph, however, lacks of the information of unclear locations. Therefore, we also provide a tabular form of the graph, including the information on missing or unclear locations. Table 2 depicts this information.

5 Evaluation Results

In this section, we report the evaluation results from the panel of tax experts (see Section 3 for more details) evaluating the software architecture description with its viewpoint that frames the concerns of tax authorities and the view of software architecture from our industry partner.

5.1 Viewpoint

Our panel confirmed the correctness of our viewpoint and agreed with the three questions distilled from the concern of international taxation. As interviewee A stated: "*I think it is possible to reveal the license structure [by your viewpoint], at least by 90%, there will always be fuzziness.*"

However, the panel also agreed that this viewpoint can only be the beginning. "*You are approaching it from a taxation perspective from a very detailed level*", as interviewee R said. "*[Your viewpoint] is an important first step towards understanding who is actually*

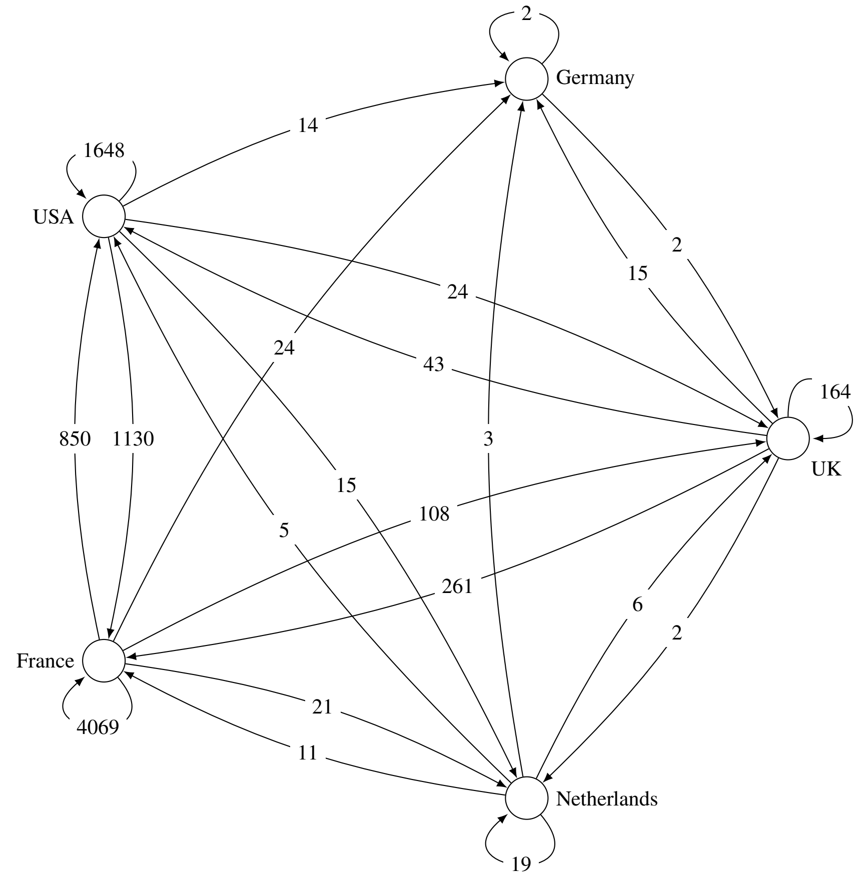

Fig. 3 The geographical distribution of the microservice architecture of our case company using our architectural viewpoint. The arrows indicate the dependency of the microservice user to the microservice owner. The countries are partially substituted by equivalent countries to ensure confidentiality for the company.

contributing to the software product. But to answer the follow-up question, which is the valuation of the software component, you need to add another layer. [Components] are not additive." In the words of interviewee K: "*Value is not counting components.*" The classification of software components potentially reflects how easily the components can be replaced or how differentiating the components from competitors are.

In this context, all participants emphasized the importance of ownership from a taxation perspective. Interviewee M and interviewee K highlighted accountability as a key aspect. They raised the question of who is actually accountable for the software product or the components from a legal perspective.

Our panel emphasized that although aggregating over jurisdictions can be acceptable in our case since there is only one subsidiary per jurisdiction, it does not generalize: If there is more than one subsidiary in a jurisdiction, aggregating over countries is not applicable. Interviewee K also mentioned that assigning legal entities to jurisdiction can become complicated referring to so-called *permanent establishments*, where, in complex situations, subsidiaries may have international branches and sites fall into jurisdiction X but belong to jurisdiction

Y. "*Assigning jurisdiction is complex. There are gray areas.*" However, those special cases are likely not relevant to those considerations.

Interviewee M wondered how often software architectures may change and commented that the viewpoint does not consider such changes over time but represents a snapshot of software architecture. As a tax auditor, the interviewee would question the representativeness of the presented snapshot.

5.2 View

When creating a view using the viewpoint, our panel identified two major issues: (1) the significant amount of unclear links between owner and jurisdiction and (2) the missing qualitative insights into the software architecture.

The first major issue refers to the problem that we were not able to identify the jurisdiction of 41.16% of all owners in our view on the given software architecture. Although the data source, the company-internal platform for teams, is promising ("*I go along with that one. [...] That is, in my opinion, an excellent data source*") since the data is not biased from a taxation perspective (interviewee M), the amount of unclear jurisdiction of owners is too high. The experts suggested different additions (jurisdiction of the team manager) or complements (a cascade of different other data sources, for example, a questionnaire sent out to all owners or geographical distribution of the team to approximate the jurisdiction of the owners). Although "*missing some percentages is no problem*", but "*half of the dependencies missing is too much*" as interviewee M explicitly said. "*50% not assignable implies that 50% of all transfer prices are not properly assigned. [...] Without data, we are in a vacuum.*" (interviewee A).

In this context, interviewee M and interviewee A emphasized the importance of traceability in tax audits. Therefore, it would also be better to reveal the process and characteristics of how teams came to their self-assignment. Indications like regular socializing in a particular country can be such an indicator. In this context, the panel agreed that the physical location of the team members and developers can be secondary from a transfer pricing perspective.

Our panel appreciated the graph representation and found it useful and utilizable. For example, interviewee M said, "*I like the [graph] visualization; it is helpful*". Interviewee A detailed that the visualization as a graph is "*helpful for a high-level view, a good starting point for looking into details*".

The tabular representation was seen as a useful extension of the graph. Interviewee M preferred the presented absolute numbers over relative numbers since absolute numbers represent the raw information and "*relative numbers are easy to calculate*" anyway.

The second major issue raised by our experts is grounded in the quantitative nature of our visualization as graph and as table. The panel of tax experts unanimously warned that the numbers may be misinterpreted as relative value added to the software product. As already discussed in the previous section, considering all components equal is not an acceptable simplification. "*The edge weight does not reflect the relevance*" as stated by interviewee R. It requires a classification of the components (interviewee K).

Besides those two major issues, the panel also raised further issues. Interviewee R wondered about the effort required to generate those insights. The interviewee emphasized that the cost-benefit ratio must be positive. Another critical remark was related to our decision to exclude non-production components. Interviewee M reminded us that our ultimate goal is to identify the value of software components. Although not in non-production or completely experimental, those components can have a significant and not negligible value for the company. Therefore, excluding them removes explanatory power from the software architecture description.

5.3 Software Architecture Description

At the end of each interview, we asked the participants about their general and high-level impression of our software architecture description with its viewpoint and view in its current state.

The overall impression was positive and, according to our panel, the software architecture description fulfilled its main purpose at its current state to a large extent. Although "*transfer pricing is not a science, it is art*", interviewee R correctly summarized our intention as follows: "*This is just fact-driven, it is no interpretation. When you ask people, they may or may not know who is involved. The nice thing about it is that it is not a subjective view; it is about facts, it is about technical details. This is what is used, if you like it or not. That is relevant because you have objectivized the contributions [to the software product]. Those are the parties involved. This allows you to have a very objective starting point to consider the potential parties you have to consider in greater detail.*" The visualization raises awareness of parties involved in the software development. The panel also confirmed the importance of software architecture for taxation: "*I think software architecture is the right perspective to understand how software is composed conceptionally*" (interviewee K). Interviewee R highlighted the need for software that can manage the complexity and support the transfer pricing processes. Interviewee M also appreciated the company's efforts in collecting and presenting the data in a tax audit: "*A company with such detailed and thorough documentation would be among the top 10 or top 5 percent of what a tax auditor usually sees.*" Interviewee K agrees: "*I really like the approach aiming for objectiveness. Let's face it: Sometimes, there are preposterous debates in tax audits.*"

6 Discussion

In this section, we critically reflect on the evaluation results and feedback from our panel of tax experts and what it means for our software architecture description and future work.

We found ambiguous assignments of owners to jurisdiction in about half of all dependencies. We expected this significant amount of unclear jurisdictions per team to create a strong headwind. Statistical estimators cannot be applied to make any assumptions on the jurisdictions because we have a strong bias: The owners with missing or unclear jurisdictions come primarily from teams with team members from different subsidiaries or are entirely virtual. Those are inherently relevant for taxation purposes and need further investigation.

The discussion on ownership, however, cast doubt on our definition of ownership. We learned that the term *ownership* triggers a particular association in the context of taxation: Tax experts think of legal ownership when they read ownership. Although we emphasized in our viewpoint that we require an owner to be accountable and responsible, the term ownership led to different misunderstandings. It triggered in-depth legal discussions that go beyond the scope of this paper. We conclude that our definition of ownership can be indeed too weak: Our understanding of accountability may fall short and does not suffice the legal accountability. Future research must address the legal aspects of code (component) ownership.

We were surprised that the quantitative nature of our description was conceived that all components are normalized and of equal importance and relevance for the software product.

Interviewee R even suggested removing the numbers completely and just indicating by the edges that there is a dependency that needs further investigation to avoid this misconception. Although we agree that those numbers can misinterpreted in such a way that all components are equally important and relevant, we still believe that the weighted edges provide value for prioritizing: The more dependencies, the more potential uncertainty is in a tax audit that needs to be addressed. To avoid the impression that dependencies are additive, we consider replacing the absolute numbers with intervals (e.g., $(0,10)$, $(10,100)$, and $(100,\infty)$) in future iterations of the software architecture description.

We are not aware of studies addressing at what rate or to what extent microservices architectures or software architecture in general change over time (a fiscal year), which also may be not generalizable and context-specific anyway. Since the effort to create the views once the data sources are identified is reasonable, we suggest regularly generating snapshots of the software architecture. Furthermore, we hypothesize that changes in software architecture also may trigger changes in ownership, directly or indirectly.

7 Limitations

In this section, we critically discuss the limitations of our study and their impact on our findings.

7.1 Anonymization

The strict anonymization of the case company required us to remove context and potentially relevant information, which included the specific business model, the revenue stream, and the organizational structure of our case company. Not being able to provide that information limits the generalizability of our evaluation. However, we believe that we provided enough context and information for the panel to provide solid and high-level feedback for the early stages of our research.

To maintain strict anonymity for the company, no participant from the company participated in the panel. The author team spoke on behalf of the company. We believe that in this early stage of the software architecture description, the author team who worked intensively with the company in other research projects and extracted the data brought enough context to the table. In future work addressing the subsequent questions of valuing software components, the involvement of internals, however, will become crucial.

We believe that anonymization has a substantial influence on the quality of the evaluation, though. We removed (intentionally) the economic context of the software architecture. Of course, this context also plays a crucial role in taxation since software engineering does not happen in the void. In particular, the economic context could not be revealed, which led to several loose ends during the evaluation. We believe that this is a major limitation of our findings since the devil is in the details. We could not investigate the potential effects of economic or managerial decisions on the software architecture.

7.2 Data Source for Software Architecture

For creating the software architecture description, we relied on historical information extracted from an internal system that continuously collects the software product's component structure (which also comprises microservices) from the code repositories. We believe that this system is a reliable and trustworthy data source for extracting the data we need since the data is also used internally for other purposes. The internal team responsible for maintaining the system helped us to avoid misinterpretation of the data. The date of extraction was randomly selected from the first two quarters in 2023. Although we did not collect this data, we assume that the microservice architecture does not change at a daily rate but is stable, at least for over weeks.

7.3 Panel Composition

Both representatives for the tax authorities come from only one country. Although the taxation standards are international and our panel consists of tax experts advising taxation internationally, our study may not fully capture the nuances in interpreting the standards. Therefore, our results only apply to the fundamentals of describing software architectures to tax authorities. Future research is required to address the nuances and country-specific specialization of international taxation.

8 Conclusion

The panel of tax experts found that our proposed architectural viewpoint properly and sufficiently frames the concerns of taxation stakeholders. Although the resulting view of the globally distributed software architecture of our industry partner has to cope with a significant amount of uncertainty, tax experts are used to working with uncertainty ("*there is no black or white in tax audits but a lot of gray*", as interviewee R summed up) and appreciate the evidence-based, empirical approach for visualizing the software architectures to tax experts as software-engineering laymen. However, we believe that we only see the tip of the iceberg of tax compliance challenges for software-developing companies.

First, we found almost 70% of all reuse relationships between the 2560 microservices in our case are cross-border and, therefore, taxable. The extent is magnitudes higher than what we would have expected. Therefore, we urge multinational enterprises to start collecting information about their software architecture and ownership continuously. Accurately collecting this information is never trivial but nearly impossible in hindsight for complex software systems. At this point, we would like to emphasize that the quality and rigor of the data curation at our case company is extraordinary and, from our experience, not industry standard. Not all companies can rely on such a solid foundation grounded in historical and accurate data on their software architectures. This uncertainty introduces further risks and lets tax audits stray away from an empirical-based, data-driven discussion on the value of software.

Second, it goes without saying, that also non-microservice architectures are subject to tax compliance. Future research is needed to discuss the challenges of other architectural styles. This may require a broader discussion that defines a software component that goes beyond the existing vague and context-specific definitions of software components (for example, (Broy et al. 1998)).

Third, as we learned in the course of this study (see Section 5), the definition of ownership in the context of software engineering and the definition of ownership in taxation overlap but are not congruent due to the different contexts and perspectives towards ownership.

Ownership in software engineering, on the one hand, refers to the assignment of responsibility and accountability for specific portions of software to individuals or teams within a development organization. There are different perspectives on ownership and autonomy in software development organizations. Software development organizations usually adopt the weak ownership principle (Fowler 2006). This principle entails assigning ownership of specific components (or microservices) to individual teams. In an ideal scenario, the designated team assumes responsibility as the primary (or exclusive) contributor to the associated component. Nevertheless, empirical analysis of real-world instances often reveals a departure from this idealized model (Borg et al. 2023; Tanveer et al. 2023; Greiler et al. 2015; Bird et al. 2011). In practice, a given component may offer support to multiple business streams, undergo development by distinct teams, and necessitate the involvement of specialized teams to address non-functional requirements, such as security and reliability. Consequently, while a specific team retains responsibility for the component, the extent of its contribution to the component can exhibit considerable variability (Zabardast et al. 2022). Tax professionals, on the other hand, begin from a civil law perspective and understand ownership in terms of the legal or economic owner regarding the right of exploitation (OECD 2022). While the identification of the legal or economic owner is relevant for balance sheet allocations, in the context of transfer pricing, the legal owner is considered (only) as the starting point for a functional and risk analysis that aims for a profit allocation commensurate with the value-adding activities performed by the legal entities within the group, the so-called *functional ownership* (OECD 2022). We believe that describing and discussing the intersection, as well as the difference in definitions, requires further interdisciplinary research. This work is not per se a purely academic exercise but an endeavor with significant practical relevance for tax compliance.

And, last but not least, many research endeavors that emerge from academia-industry collaborations are prone to the challenge that not all data can be shared “as-is” for confidentiality reasons. We would normally argue for, at least, partial disclosure, anonymization of disclosed data, or sharing of metadata (and, where reasonable, making data conditionally available upon request). For this research line, however, absolute anonymity and confidentiality between the researchers and all involved parties are paramount to foster trust, honesty, and candor among the participants to ensure that individuals freely contribute highly sensitive information without fear of identification or reprisal. While only an absolute dedication to the anonymity of our case and all involved subjects can safeguard confidentiality and privacy, it also poses a natural barrier to any data dissemination and, more generally, to open science, which we ourselves hold dear as the de-facto standard for empirical software engineering (Mendez et al. 2020). In our line of research, however, we are simply not able to reveal any details about the company, the software architecture, or the interviews without potentially revealing the identity of the involved parties. Even how we collected the data or what data source we used would pose the risk, even if remotely, of revealing the identity of the participating company. This constitutes a fundamental barrier to reproducibility, also in light of that we deem a replication with publicly available data not feasible nor ethical (also considering that we can also not make a case for using open source projects which we discussed already to be not comparable). At the same time, we may well argue that our reporting is still suitable to uncover the potential of describing globally distributed software architectures for the purpose of critically analyzing tax compliance and, at the same time, more fundamental questions that open an interesting avenue toward tax compliance in software engineering. In any case, this general dilemma still remains, in our view, something we deem important to discuss in the research community of empirical software engineering.

Data Availability

Due to the nature of the research, supporting data, except interview guidelines[2], is not available. Please see our discussion in Section 8.

Conflict of Interest

The authors declare that they have no conflict of interest.

Acknowledgement

We would like to thank our industry partner and the panel of tax experts for their great support, which made this research possible. This work was supported by the KKS Foundation through the SERT Project (Research Profile Grant 2018/010) at Blekinge Institute of Technology.

References

Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don't touch my code! examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, pp 4–14

Borg M, Tornhill A, Mones E (2023) U owns the code that changes and how marginal owners resolve issues slower in low-quality source code. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, pp 368–377

Broy M, Deimel A, Henn J, Koskimies K, Plášil F, Pomberger G, Pree W, Stal M, Szyperski C (1998) What characterizes a (software) component? Software - Concepts & Tools 19:49–56, DOI 10.1007/s003780050007

Buchner S, Riehle D (2022) Calculating the costs of inner source collaboration by computing the time worked. In: Proceedings of the 55th Hawaii International Conference on System Sciences, HICSS'22, DOI 10.24251/HICSS.2022.896

Capraro M (2020) Measuring inner source collaboration. doctoralthesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Cataldo M, Herbsleb JD (2008) Communication networks in geographically distributed software development. In: Proceedings of the 2008 ACM conference on Computer supported cooperative work, ACM, pp 579–588, DOI 10.1145/1460563.1460654

Cataldo M, Herbsleb JD, Carley KM (2008) Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, pp 2–11, DOI 10.1145/1414004.1414008

Clarke V, Braun V (2017) Thematic analysis. The Journal of Positive Psychology 12:297–298, DOI 10.1080/17439760.2016.1262613

Clerc V, Lago P, van Vliet H (2007) Global software development: Are architectural rules the answer? In: International Conference on Global Software Engineering (ICGSE 2007), IEEE, pp 225–234, DOI 10.1109/ICGSE.2007.21

Conway ME (1968) How do committees invent. Datamation 14(4):28–31

Dorner M, Capraro M, Treidler O, Kunz T, Smite D, Zabardast E, Mendez D, Wnuk K (2024) Taxing collaborative software engineering. IEEE Software (01):1–8, DOI 10.1109/MS.2023.3346646

Fowler M (2006) Codeownership. URL `https://martinfowler.com/bliki/CodeOwnership.html`

Greiler M, Herzig K, Czerwonka J (2015) Code ownership and software quality: A replication study. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, IEEE, pp 2–12

Herbsleb J, Grinter R (1999a) Architectures, coordination, and distance: Conway's law and beyond. IEEE Software 16:63–70, DOI 10.1109/52.795103

[2] `https://zenodo.org/records/10214942`

Herbsleb J, Mockus A (2003) An empirical study of speed and communication in globally distributed software development. IEEE Transactions on Software Engineering 29:481–494, DOI 10.1109/TSE.2003.1205177
Herbsleb JD, Grinter RE (1999b) Splitting the organization and integrating the code. In: Proceedings of the 21st International Conference on Software Engineering, ACM, pp 85–95, DOI 10.1145/302405.302455
ISO/IEC/IEEE42010:2022 (2022) Software, systems and enterprise — Architecture description. Standard, International Organization for Standardization, Geneva, CH
Kwan I, Schroter A, Damian D (2011) Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. IEEE Transactions on Software Engineering 37:307–324, DOI 10.1109/TSE.2011.29
Mendez D, Graziotin D, Wagner S, Seibold H (2020) Open Science in Software Engineering, Springer International Publishing, pp 477–501. DOI 10.1007/978-3-030-32489-6_17
Newman S (2021) Building microservices. O'Reilly Media, Inc.
OECD (2015) Addressing the Tax Challenges of the Digital Economy, Action 1 - 2015 Final Report. DOI 10.1787/9789264241046-en, URL https://www.oecd-ilibrary.org/content/publication/9789264241046-en
OECD (2022) OECD Transfer Pricing Guidelines for Multinational Enterprises and Tax Administrations 2022. DOI 10.1787/0e655865-en, URL https://www.oecd-ilibrary.org/content/publication/0e655865-en
Olbert M, Spengel C (2017) International taxation in the digital economy: Challenge accepted? World tax journal 9(1):3–46
Ruhe G, Wohlin C (eds) (2014) Software Project Management in a Changing World. Springer Berlin Heidelberg, DOI 10.1007/978-3-642-55035-5
Stol KJ, Fitzgerald B (2018) The abc of software engineering research. ACM Transactions on Software Engineering and Methodology 27:1–51, DOI 10.1145/3241743, URL http://dl.acm.org/citation.cfm?doid=3276753.3241743
Tanveer B, Zabardast E, Gonzalez-Huerta J (2023) An approach to align socio-technical dependencies in large-scale software development. In: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), IEEE, pp 341–347
Treidler O, Kunz TE, Capraro M, Dorner M (2024a) Sustaining arm's length cost allocations for highly integrated development functions: An explorative case study of transfer pricing for innersource communities. Tech. Rep. 2024:01, Blekinge Institute of Technology, Department of Software Engineering
Treidler O, Kunz TE, Dorner M, Capraro M (2024b) The microsoft case: Lessons for post-beps software development cost contribution arrangements. Tax Notes International 114:1883
Tureček T, Šmiřák R, Malík T, Boháček P (2010) Energy Project Story: From Waterfall to Distributed Agile, Springer Berlin Heidelberg, pp 362–371. DOI 10.1007/978-3-642-13054-0_39
Zabardast E, Gonzalez-Huerta J, Tanveer B (2022) Ownership vs contribution: Investigating the alignment between ownership and contribution. In: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), IEEE, pp 30–34
Šmite D (2014) Distributed Project Management, Springer Berlin Heidelberg, pp 301–320. DOI 10.1007/978-3-642-55035-5_12
Šmite D, Moe NB, Floryan M, Gonzalez-Huerta J, Dorner M, Sablis A (2023) Decentralized decision-making and scaled autonomy at spotify. Journal of Systems and Software 200:111649, DOI 10.1016/j.jss.2023.111649