Taxing Collaborative Software Engineering

Michael Dorner, *Blekinge Institute of Technology, Sweden*

Maximilian Capraro, *Kolabri, Germany*

Oliver Treidler, *Kolabri, Germany*

Tom-Eric Kunz, *Kolabri, Germany*

Darja Smite, *Blekinge Institute of Technology, Sweden*

Ehsan Zabardast, *Blekinge Institute of Technology, Sweden*

Daniel Mendez, *Blekinge Institute of Technology, Sweden and fortiss, Germany*

Krzysztof Wnuk, *Blekinge Institute of Technology, Sweden*

Abstract—The engineering of complex software systems is often the result of a highly collaborative effort. However, collaboration within a multinational enterprise has an overlooked legal implication when developers collaborate across national borders: It is taxable. In this article, we discuss the unsolved problem of taxing collaborative software engineering across borders. We (1) introduce the reader to the basic principle of international taxation, (2) identify three main challenges for taxing collaborative software engineering making it a software engineering problem, and (3) estimate the industrial significance of cross-border collaboration in modern software engineering by measuring cross-border code reviews at a multinational software company.

INTRODUCTION

> *"He's spending a year dead for tax reasons."*
>
> *– Douglas Adams, The Hitchhiker's Guide to the Galaxy*

Modern software systems are too large, too complex, and evolving too fast for single developers to oversee. Therefore, software engineering has become highly collaborative. Further, software development is often a joint effort of individuals and teams collaborating across borders, especially in multinational companies with their subsidiaries spread around the globe.[1] However, **collaboration has a legal implication if individuals collaborate across borders: the profits from those cross-border collaborations become taxable.**

In this article, we describe the complexity of applying the established international taxation standards required and enforced by national tax authorities in the context of modern software engineering with its distributed and fine-grained collaboration crossing borders. We start with a gentle introduction to international standards in multinational taxation and its basic *arm's length principle* for software engineers. We then discuss the challenges of taxing collaborative software engineering and illustrate the industrial significance of cross-border collaboration in an industrial case, namely code review.

Taxation in software industry has been debated for many decades.[2] 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 and broad discussion.[3] Here, we extend the debate to software engineering, the way the software products and services are being developed, which has not yet been covered. Our goal is to raise a debate and draw attention to this problem among a software engineering audience. For in-depth information on basic transfer pricing concepts including standard methods and tax compliance requirements, we recommend the interested reader further readings.[4]

A GENTLE INTRODUCTION TO TAXING MULTINATIONAL ENTERPRISE FOR SOFTWARE ENGINEERS

Consider *devnullsoft Group*, a multinational enterprise that develops and sells software-intensive product, has two legal entities: *devnullsoft GmbH* in Germany and its subsidiary *devnullsoft AB* in Sweden. The German development team employed by *devnullsoft GmbH* develops the software-intensive product jointly with the Swedish development team employed by the Swedish subsidiary *devnullsoft AB*. 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 tax authorities are left out in the cold because *devnullsoft AB* has no share of the profit that could be taxed in Sweden, although *devnullsoft AB* contributed significantly to the product through code contributions, code reviews, bug reports, tests, architectural decisions, or other contributions 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*.[5]

The *arm's length principle* is 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. This principle ensures that transfer prices between associated companies of multinational enterprises are established on a market value basis and not misused for profit shifts from high to low tax regions.

To comply with the arm's length principle, *devnullsoft GmbH* in Germany and *devnullsoft AB* in Sweden need to operate from a taxation perspective as if they were not associated. Since a regular participant in the market would not provide code contributions, code reviews, tests, or architectural designs free or other contributions of charge to a closed-source software project, *devnullsoft GmbH* in Germany needs to pay for the received contributions, the so-called *transfer price*.

Transfer prices are the prices at which an enterprise transfers physical goods and intangibles or provides services to associated enterprises. Since software is intangible itself, the transfer of intangibles like source code, code reviews, bug reports, etc., is our focal point. This transfer price guarantees that *devnullsoft AB* gets its share of the profit, which then can be taxed by the Swedish tax authorities.

In Figure 1, we provide a schematic overview of transfer pricing between the two associated software companies from our example. Although *devnullsoft AB* contributed significantly to the software-intensive product, without a transfer price, *devnullsoft AB* has no share of profits; all profits are fully taxable in Germany only. However, if *devnullsoft GmbH* in Germany pays a transfer price reflecting the value for the services and intangible properties received from its Swedish associated enterprise, *devnullsoft AB* realizes profits which then are taxable in Sweden.

In our case, the *devnullsoft Group* does not artificially shift profits to a tax haven. Yet, one can easily imagine that neglecting to charge arm's length prices can be intentionally misused for profit-shifting. Therefore, the OECD guidelines permit tax authorities like the Swedish tax authority to adjust the transfer price where the prices charged are outside an arm's length range. Such an adjustment will carry interest and might be coupled with penalties. In the wake of the OECD's *Base Erosion and Profit Shifting (BEPS) Project*[a], the regulatory framework has become considerably stricter at an international and national level. As a result, tax authorities can demand more comprehensive information to detect misalignments and enforce tax adjustments. From the companies' perspective, its software development may be—intentionally or unintentionally—non-compliant and face the risk of being legally prosecuted.

CHALLENGES

So, what are transfer prices for collaborative software engineering that comply with this arm's length principle? Determining a market price for intangibles is inherently difficult and is reflected in a broad price range. Collaborative software engineering, however, scales the problem of a transfer-price determination to a new level of complexity because the reality of modern software engineering is significantly more complex than our introductory example above may suggest. Since transfer price regulations apply to a much broader definition of intangibles compared to accounting standards, the latter can not be used as a reliable measure of value for transfer pricing purposes.[5]

In the following, we discuss three main high-level challenges for transfer pricing in collaborative software engineering within multinational enterprises. Figure 2
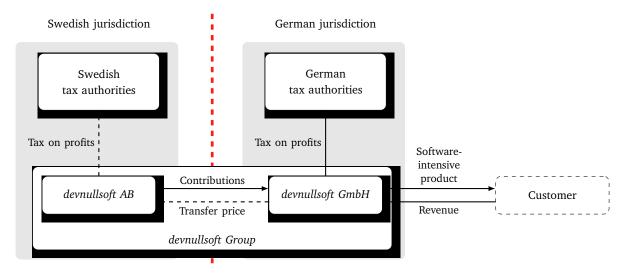
---

[a]https://www.oecd.org/tax/beps/

FIGURE 1. A schematic overview of the necessity and mechanics of transfer pricing in a multinational enterprise (*devnullsoft Group*) with two associated software companies (*devnullsoft AB* in Sweden and *devnullsoft GmbH* in Germany): Without considering a market-based compensation, the so-called *transfer price*, *devnullsoft AB* has no share on the profits which could be taxed by the Swedish tax authorities; all profits are with *devnullsoft GmbH* and, therefore, all taxes stay within Germany.

highlights the complexity in modern collaborative software engineering at *devnullsoft Group* and where those three challenges apply.

Challenge 1: What is a taxable transaction in software engineering?

The trouble for transfer pricing in software engineering begins with a fundamental question: What is actually a taxable transaction in the context of collaborative software engineering? We simply do not know what types or characteristics types or characteristics classify a taxable exchange of intangibles or services across the boundaries of a country in the context of software engineering.

Among other potentially relevant types of taxable intangibles, such as *goodwill* or *group synergies*, we discuss in this and the following subsections two types of intangibles that are highly relevant for software engineering: know-how and licenses.

The OECD Transfer Pricing Guidelines define **know-how** as the "proprietary information or knowledge that assist[s] or improve[s] a commercial activity, but that [is] not registered for protection in the manner of a patent or trademark". The commercial activity includes the manufacturing, marketing, research and development of and for a software system.

Does the OECD definition imply that all types of information exchanged during collaborative software engineering are know-how? On the one hand, yes, since all information is proprietary and, to some extent, con-

tributes to the software being developed or its engineering processes. But on the other hand, how do we know which information assists or improves the commercial activity, meaning the engineering of the software system, over time? For example, a quick and dirty bug fix without sufficient documentation or testing may improve the software system in the present but makes changes more costly or even impossible in the future. Making such sub-optimal decisions leads to incurring technical debt,[6] which is potentially relevant for taxation. Thinking the concept of technical debt ahead, we have begun to understand that similar to physical, tangible assets, software assets degrade and lose value inevitably due to intentional or unintentional decisions caused by technical or non-technical manipulation of the asset or associated assets during all stages of the product life-cycle.[7] Such an asset degradation will also be of great interest from a taxation perspective.

The second type of intangibles highly relevant to transfer pricing in software engineering if transferred across borders is **licenses**. Although maybe not even explicit, the company-internal use and reuse of components is an instance of licensing. Complex software systems are not monolithic blocks of code but consist of components that are developed, shared, and reused by separate teams. However, we lack a common understanding of software components and reuse in software engineering for taxation. Not every component is directly used for or in a software-intensive product, but maybe adds value to the product. For example, a well-engineered
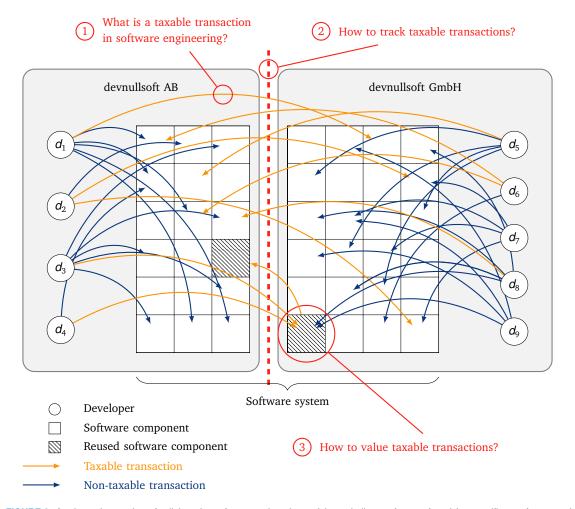
FIGURE 2. A schematic overview of collaborative software engineering and three challenges for transfer pricing specific to software engineering.

CI/CD pipeline accelerates the development cycles and brings new features or bug fixes faster to the customers.[8] Furthermore, it is also not always clear who owns, contributes to, or uses a component within a company, and the roles may even change over time.[9] In contrast to open source, the reuse is often implicit, lacking a company-wide license agreement that clarifies the responsibility and accountability between component owners and users. Even worse, we do not even know if our definition and understanding of code ownership[10] suffices the definition of ownership in a taxation context.

Additionally, we see an interplay between those two types of intangibles, know-how and licensing, since they may be two sides of the same collaboration. For example, when code contributions from the component user support instance of reuse.

> **Insight**
>
> Identifying the taxable transactions requires either a holistic perspective of software engineering or at least suitable, practical, and accurate proxies. Compliant software engineering needs a common understanding and a taxonomy of taxable transactions specific to software engineering.

Challenge 2: How to track cross-border transactions in software engineering?

Also, the practical tracking of taxable know-how and licensing (and potentially other types of intangibles) is a challenge on its own.

Tracking know-how is an inherently difficult task. Since the teams collaborating are no longer colocated, numerous tools enable an exchange of know-how in

software engineering. Those tools are suitable as rich data sources to different extents: While domain-specific tools like issue trackers or collaborative software development platforms like GitHub or Gitlab often track the exchanges very thoroughly, other communication and collaboration tools do not: Online meetings, for example, can facilitate an exchange of taxable intangibles, but this exchange is not tracked by any tool. But even if there is a rich data basis available, leveraging those data sources is problematic for following reasons:

› *Establishing location*—It can be difficult to establish the location of collaborators or capture when a location of a collaborator has changed, because organizations often preserve only the latest version of the organizational structures.
› Privacy—Analyzing the complete communication of developers may be perceived as a measure of surveillance, which raises ethical and legal concerns related to privacy.

In contrast to the potentially rich sources for tracing taxable transactions from collaboration tools used in software engineering, tracking company-internal reuse often lacks a solid data basis. Although companies often track the reuse of external open-source components for open-source license compliance purposes, those tools are rarely used or suitable for tracking company-internal reuse. Also, only reuse that crosses borders is taxable; information that is often not available or stored over time—although component ownership is not static and may be subject to change.

Insight

Data for tracking taxable transactions may be incomplete, faulty with respect to location, or restricted. There is no dedicated tool support yet for the practical transfer price determination.

Challenge 3: How to value taxable transactions in software engineering?

While it is inherently difficult to tax intangibles in general, things are even more complicated in software engineering. Potentially taxable intangibles cover a large range of granularity in software engineering: They may be as large as a microservice providing user authentication used by microservices of other teams (→ intangible *licenses*) or as small as a code change, code review, or bug report (→ intangible *know-how*). Although the code change or feedback in a code review is small—maybe even only one line of code, like in the case of the *Heartbleed* security bug in the OpenSSL cryptography library from 2014[11]—the potential impact on the software system can be tremendous or even fatal. A software change or a code review delivers value through impact, not size.

The same applies to licensing. The number of use relations of a software component or its size (however defined) does not reflect the value provided to the software-intensive product. While the software component for user authentication may be important for operating the software-intensive product and, therefore, has a large amount of depedent software components, it is not differentiating and may even be considered a commodity.

This means we cannot simply use purely quantitative measurements for transfer pricing. However, the sheer mass of small, fine-grained transactions of all types makes a human qualitative case-by-case evaluation impossible.

Insight

A purely quantitative valuing can hardly reflect the value of transactions; however, a purely qualitative assessment does not scale with the magnitude of cross-border transactions in modern software development.

AN INDUSTRIAL EXAMPLE OF CROSS-BORDER COLLABORATION

So, is cross-border collaboration and, therefore, also the taxation of it, a real issue? To estimate the prevalence of cross-border collaboration, we measure cross-border code reviews as proxy for cross-border collaboration in a typical industrial setting.

A cross-border code review is code review with participants from more than one country. Although it originated in collocated, waterfall-like code inspections, its modern stances are lightweight and asynchronous discussions among developers around a code change. Different tools are in use, for example, Gerrit or Github and Gitlab with their implementation of code review as so-called pull and merge requests. Although code review is by far not the only type of collaboration that may include taxable transactions and is also likely not sufficient to determine company-wide transfer prices, the following characteristics make code review a suitable first proxy for cross-border collaboration:

› More than code only—Code review not only includes the actual code change and its authors but also includes the feedback from reviewers that may have formed or changed the code change significantly but is no longer visible in the repository after merging the code change into the code base.
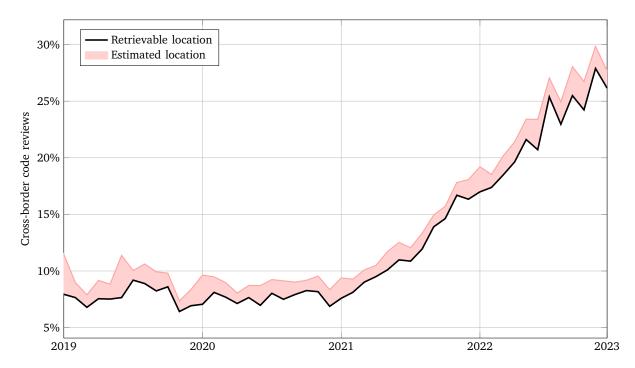
FIGURE 3. The share of cross-border code review at our case company in the years 2019, 2020, 2021 and 2022 (black line) monthly sampled. Since not all historical locations of all code review participants could be reliably retrieved, the share of cross-border reviews could be more significant (indicated by the red area).

Therefore, our proxy goes beyond existing code-based measurements for collaboration.[12]

› Accessible & complete—The code review discussions are (company-internally) public by default and are, thus, accessible. Unlike other tools like instant messaging services or e-mail, code review does not split into public and private, whose analysis may cause privacy concerns.

› Persistent—Code review tools are the backbone of modern code review and ease data extraction. Other types of code review (for example, private or synchronous discussion around a code change through meetings or instant messaging) may not be captured through the tooling, though.

We measured the share of cross-border code reviews at a multinational company delivering software and related services worldwide with main R&D locations in three countries. For many years the company has tried to allocate products to particular sites to avoid the burden of cross-border collaboration. However, our analysis shows that developers represent more than 25 locations because the new corporate work flexibility policy permits relocations.[13] The company uses a single, central, and company-wide tool for its internal software development and code review. Understandably, our case company

wants to remain anonymous. Therefore, we are not able to describe the case any further. However, we believe that our case company is exemplary for a multinational enterprise developing software.

From the code review tool, we extracted all code reviews that were completed in 2019, 2020, 2021, and 2022 including their activities. All bot activities were removed and were not considered in our analysis. We then modelled code reviews as communication channels among code review participants.[14] We consider a code review as a discussion thread that is completed as soon as no more information regarding a particular code change is exchanged (i.e., the code review is closed). We complement each code review participant with the information of the country of the employing subsidiaries at the time of the code review.

We provide a replication package to reproduce our results for any GitHub Enterprise instance[b]. Due to the sensitive topic, we are not able to share our data.

Figure 3 shows an increase in relative cross-border code reviews over time. The share of cross-border code reviews was between 6% and 10% in 2019 and 2020. Yet, we see a further steep increase reaching between

---

[b]See https://github.com/michaeldorner/tax_se

25% and 30% at the end of 2022.

Interestingly, 6% of all cross-border code reviews involve participants from more than two countries. This means transfer pricing in collaborative software engineering becomes not only a bilateral but a multilateral problem with not only two but multiple—in our case company up to six—different jurisdictions and tax authorities involved in the transfer pricing process.

Although the share of cross-border collaboration may vary between companies, yet, our findings suggest that—through the proxy of cross-border code reviews—cross-border collaboration becomes a significant part of daily life in multinational software companies. It is fair to assume that a further increase in cross-border collaborations in software engineering will draw the attention of tax authorities.

CONCLUSION

On the one hand, the arm's length principle is the de-facto standard for multinational enterprises that any multinational company must comply with. On the other hand, software engineering is highly collaborative—beyond geographical and organizational boundaries. Determining a reasonable transfer price for this cross-border collaboration brings the general challenge of taxing intangibles to a new level of complexity.

Pretending to be dead for tax reasons is no option because ignoring the significant cross-border collaboration in modern software development, as we exemplarily found, is a slippery slope: Cross-border collaborations in software engineering will draw the attention of tax authorities. Also, ceasing or forbidding all cross-border collaboration in software engineering is not a valid solution: Reversing the collaborative nature of modern software engineering is likely too costly and takes too long.

Obviously, neither there are simple solutions for such a complex and interdisciplinary problem, nor a single article can solve this complex problem potentially affecting every software-developing company with developers employed by subsidiaries in more than one country. However, our article aims to bring this eminent and unsolved problem of taxing collaborative software engineering to the audience that can solve this issue. As a software engineering community, we will need to find a common understanding of what constitutes taxable transactions and each company that develops software collaboratively within more than one country needs to learn how to track and value cross-border collaboration, how to estimate the transfer pricing, and how to report these to the tax authorities to be compliant.

References

1) J.D. Herbsleb and D. Moitra. "Global software development". In: *IEEE Software* 18 (2 2001), pp. 16–20. DOI: 10.1109/52.914732.

2) OECD. *Addressing the Tax Challenges of the Digital Economy, Action 1 - 2015 Final Report*. 2015, p. 288. DOI: 10.1787/9789264241046-en.

3) Marcel Olbert and Christoph Spengel. "International Taxation in the Digital Economy: Challenge Accepted?" In: *World tax journal* 9.1 (2017), pp. 3–46.

4) Oliver Treidler. *Transfer Pricing in One Lesson*. Springer International Publishing, 2020. DOI: 10.1007/978-3-030-25085-0.

5) OECD. *OECD Transfer Pricing Guidelines for Multinational Enterprises and Tax Administrations 2022*. 2022, p. 659. DOI: 10.1787/0e655865-en.

6) Nicolli Rios, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners". In: *Information and Software Technology* 102 (Oct. 2018), pp. 117–145. DOI: 10.1016/j.infsof.2018.05.010.

7) Ehsan Zabardast et al. "Assets in Software Engineering: What are they after all?" In: *Journal of Systems and Software* 193 (Nov. 2022), p. 111485. DOI: 10.1016/j.jss.2022.111485.

8) Brian Fitzgerald and Klaas-Jan Stol. "Continuous software engineering: A roadmap and agenda". In: *Journal of Systems and Software* 123 (Jan. 2017), pp. 176–189. DOI: 10.1016/j.jss.2015.06.063.

9) Ehsan Zabardast, Javier Gonzalez-Huerta, and Binish Tanveer. "Ownership vs Contribution: Investigating the Alignment Between Ownership and Contribution". In: IEEE, Mar. 2022, pp. 30–34. DOI: 10.1109/ICSA-C54293.2022.00013.

10) M.E. Nordberg. "Managing code ownership". In: *IEEE Software* 20 (2 Mar. 2003), pp. 26–33. DOI: 10.1109/MS.2003.1184163.

11) Marco Carvalho et al. "Heartbleed 101". In: *IEEE Security & Privacy* 12 (4 July 2014), pp. 63–67. DOI: 10.1109/MSP.2014.66.

12) Maximilian Capraro, Michael Dorner, and Dirk Riehle. "The patch-flow method for measuring inner source collaboration". In: ACM Press, 2018, pp. 515–525. DOI: 10.1145/3196398.3196417.

13) Darja Smite et al. "Work-from-home is here to stay: Call for flexibility in post-pandemic work policies". In: *Journal of Systems and Software* (Jan. 2022), p. 111552. DOI: 10.1016/j.jss.2022.111552.

14) Michael Dorner et al. "Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs". In: *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '22. Helsinki, Finland: Association for Computing Machinery, 2022, pp. 195–204. DOI: 10.1145/3544902.3546254.