

Describing Globally Distributed Software Architectures for Taxation

Before we start

Before we start

- Please ask (interrupt) me any time!
- Please be honest and blunt!
- This interview is completely **confidential** and **anonymous**!
- May we record this interview?

About you

- What is your current position?
- How long have you been working in the context of transfer pricing?
- Do you have contact points with software (companies) or software-intensive products and their taxation? Do you know someone who has?
- Let's assume there are two perspectives on taxation: A tax payer perspective (companies) and a tax auditor perspective (tax authorities, government). Which perspective do you have?

Gentle introduction to modern software engineering

Software as a cookbook

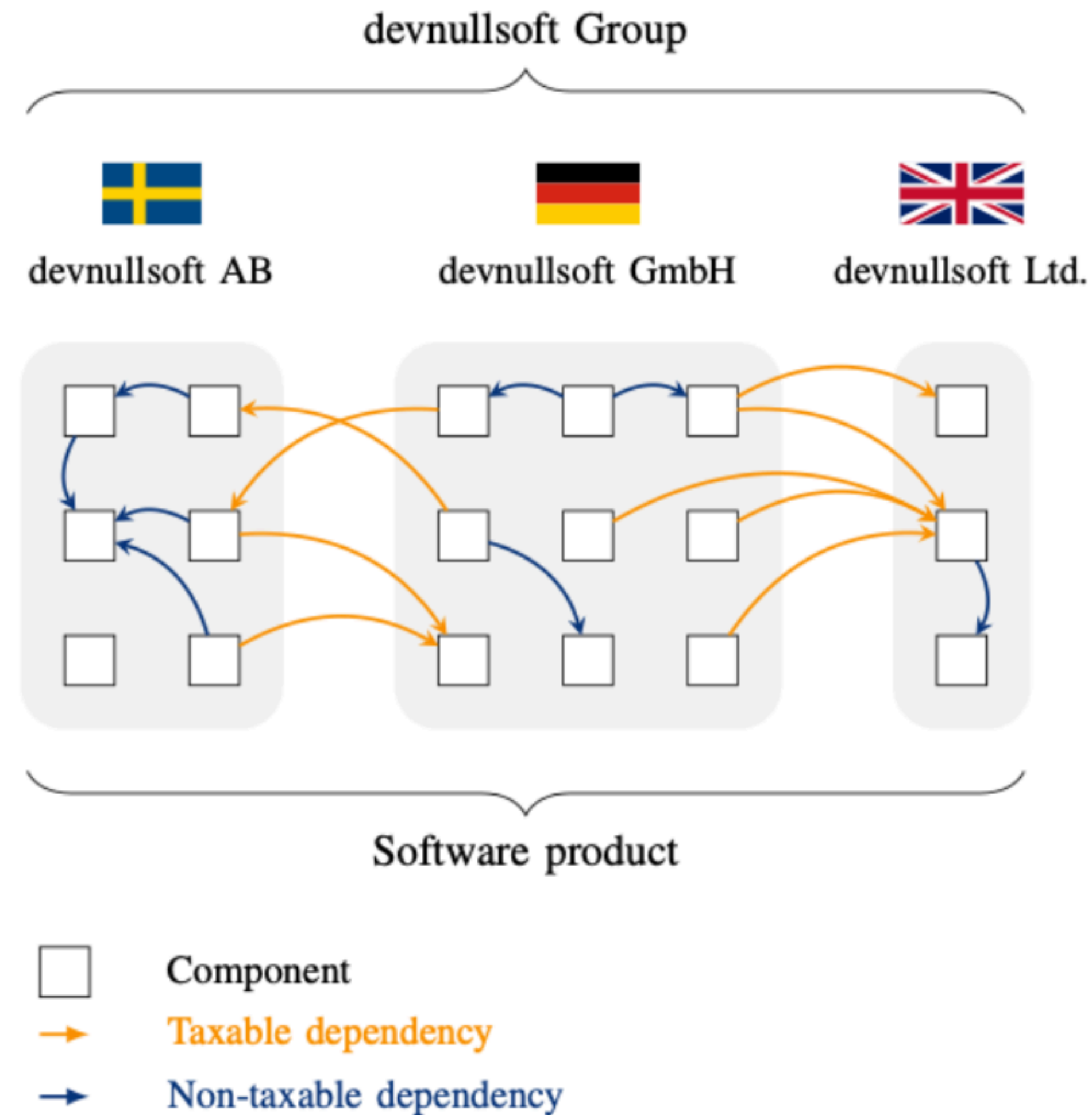
- Set of instructions
 - Some are trivial, other are complex
 - Ingredients in, food out on different layers
- For different audiences
 - cookbook for running a restaurant or for cooking at home
- Continuously improved and changed
- Multiple authors
- Reuse of smaller instruction sets (e.g., for a delicious broth)



What is a software architecture?

- Software architecture
 - is the high-level structure of a software system and how its components interact with each other
 - provides a blueprint for designing and implementing software systems

Globally distributed software architectures



The three questions we need to answer

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

The evaluation

Goal and context

- The main question of our study is:
How can multinational enterprises describe their globally distributed software architectures to tax authorities?
- We propose such a software architecture description for tax audits
- This interview aims to evaluate its quality, completeness, and utilizability in its current state



Two evaluation aspects

- **Framing** a software architecture description for taxation (*viewpoint*)
- **Applying** a software architecture description in a real-world case (*view*)

Framing a software architecture
description for taxation

Four steps for a software architecture description

1. Defining component and component owner
2. Decomposing the component structure of the software product
3. Identifying components owners
4. Identifying jurisdictions of component owners

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. Software components can take many forms and sizes depending on the abstraction level.
- **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. Only one organizational unit is 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.**

2. Decomposing the component structure of the software product

To reveal the component structure and the dependencies of software components, **the software product must be decomposed in the component structure.** 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.

3. Identifying components 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. 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.

Questions

- Do you think such a description represents what you actually need in a tax audit?
 - What is missing?
 - What is unclear?
 - What did you expect differently?
 - What is too much or unnecessary?

Applying a software architecture description in a real-world case

Context

- All data are collected from a multinational enterprise developing a large web-based platform.
- We describe in the following how we extracted and aggregated the data according to our definitions.

1. Defining component and component owner

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. Therefore, it meets the definition of a software component.

Each microservice is usually owned by a single team. We excluded 42 microservices (1.64%), each owned by one individual, because the locations of individuals were not accessible for this research due to privacy concerns.

2. Identifying components owners

There are 336 different teams owning microservices.

3. Decomposing the component structure of the software product

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

2,518 microservices have 16,533 dependencies between them.

4. Identifying jurisdictions of component owners

The geographical locations of the teams are self-assigned: Teams report their location in a central system. We find this most realistic in the world of distributed and remote teams. However, we were not able to retrieve the geographical location of 140 teams (41.16%) with this 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 8,097 use relationship, i.e., 48.94% of all use relationships.

Aggregation over countries

- Due to the large numbers, we aggregated over the jurisdiction of subsidiaries (in our case there is only one subsidiary per country)
- Next slide: visualizing the aggregation
 - as **graph** (without missing locations) and
 - as **table** (with missing locations)

Questions

- How helpful or unhelpful is the aggregation over countries?
- How helpful or unhelpful is the visualization as a graph for interpreting the software architecture in the context of taxation?

		Component owner					
		DEU	GBR	NLD	SWE	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
	SWE	24	108	21	4069	850	1767
	USA	14	24	15	1130	1648	642
	N/A	27	70	14	2283	970	2171

Questions

- How helpful or unhelpful is the visualization as a table for interpreting the software architecture in the context of taxation?

Questions

- How comfortable would you feel to use our view in a tax audit?
- Is there anything we haven't talked about yet or you would like to let us know about?

THANK YOU I SAY



GRATEFUL I AM