Calculating the Costs of Inner Source Collaboration by Computing the Time Worked

Stefan Buchner
Computer Science Department
Friedrich-Alexander University
Erlangen-Nürnberg, Germany
stefan.buchner@fau.de

Dirk Riehle
Computer Science Department
Friedrich-Alexander University
Erlangen-Nürnberg, Germany
dirk@riehle.org

Abstract

A key part of taxation, controlling, and management of international collaborative programming workflows is determining the costs of a supplied software artifact. The OECD suggests the use of the Cost Plus method for calculating these costs. However, in the past, this method has been implemented using only coarse-grain data from the costs of whole organizational units. Due to the move to inner source software development, we need a much more fine-grain solution for computing the detailed time spent on programming specific components. This is necessary, because a more accurate work time distribution is required to fulfill the fiscal and administrative challenges posed by collaborating across organizational boundaries. In this article, we present a novel method to determine the time spent on an individual code contribution (commit) to a software component for use within cost calculation, especially for taxation purposes. We demonstrate the usefulness of our approach by application to a real-world data set gathered at a large multi-national corporation. We evaluate our work through feedback received from this corporation and from the German Ministry of Finance.

1. Introduction

In software developing companies, engineering managers, the finance department, and human resources all would like to know how long programming tasks took in the past and may take in the future. The reasons are manifold: An engineering manager may need this information to develop a project plan, the finance department may need this information to calculate the cost of a software component, and the human resources department may want to know this information to determine performance.

Today's common solution is to let software developers self-declare by way of checking-in and checking-out of certain work tasks, or simply by filling in time-sheets. The introduction of inner source software development by software vendors as a software development approach that complements and extends existing practices has made this insufficient.

Inner source software development is the use of open source best practices inside companies. No open source software is being developed, only its practices are being used. For this, departments open up their code base to the whole company, typically on an internal forge like GitHub, advertise their components, welcome visitors, and engage with them in the hope that such visitors will find these components useful. The goal is to get visitors to start using a component so that they will eventually contribute to it, which leads to cost reduction to both the original developers and the visitor-turned-user-turned-collaborator [1]. Other demonstrated benefits of such inner source collaboration are developing higher quality software components within the company, better knowledge sharing, and higher employee satisfaction, among others [2][3][4].

In this new world of inner source, developers not only work on tasks that have been assigned to them by their managers, they also work on and contribute to software components across the organization, often crossing organizational boundaries, even tax boundaries. The number of components a developer contributes to can increase significantly. In this situation, it becomes impossible for a developer to precisely track how they spend their time for programming different components. The number of possible components they contribute to is too large for all practical purposes. It would be better if the calculation of time spent on programming a particular code contribution could be automated.

Being able to measuring time spend on commits and assigning an economic value (e.g. costs) to it helps companies mastering their financial challenges posed by new cross-boundary software development. A solution which is able to calculate the costs of inner source might also be helpful for various accounting and profit calculations (where costs play a central role, see [5]) as well as management related challenges (e.g.

To be published as: 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. Maui, US.

product management [4] [6], risk management [7] or KPI calculation [8]). Prospectively, economic inner source assessment can help companies to manage and introduce new organization forms, business processes, information systems and developing their business overall.

In this article, we present a method to compute the time spent on programming a particular code contribution (commit to a code repository) for usage within cost calculation. It is robust towards the developer switching gears and contributing to multiple different components in short sequence.

With inner source being so closly related to economic assessment and business processes, we asked the following research question:

RQ: How can we calculate the time spend on code contributions for usage within various cost related business processes

We take a design science approach and motivate our work through the use case of calculating transfer prices for inner source contributions in globally distributed software development for a client/supplier relationship.

The remainder of this article is structured as follows: In section 2 we present related work, in section 3 we present our research design, in section 4 we identify the problem and define a research objective, in section 5 we present a solution design and its implementation, and in section 6 we first demonstrate and then evaluate our solution. In section 7 we discuss the research limitations and in section 8 we present our final conclusions.

2. Related Work

Our algorithm connects the topics inner source software development and transfer pricing. Therefore, we review research on these topics as related work.

2.1. Transfer pricing

Even though our solution might be applicable to large variety of cost calculation purposes in management and controlling, the main motivator when designing the algorithm was to calculate transfer prices, especially for taxation purposes.

While in an ordinary market, two market participants determine the price of a product or (intellectual) property in orientation to the overall market, for transferring property within one organisation, no such market exists. However, as those transactions are often performed between tax boundaries, standards for calculating the so called transfer price exist to ensure fair pricing and taxation according to the function and risk of each transfer [9] [10].

As an international standard, the OECD defined five standard methods for calculation of transfer prices [9]: The Comparable Uncontrolled Price Method, Resale Price Method, Cost Plus method as Traditional Transaction Methods or Transactional Net Margin method and Transactional Profit Split Method as Transactional Profit Methods. The exact method used is determined by the function, risk and overall situation a transaction is situated in [11].

With the growing importance of software and digital business in the economy, problems in transfer pricing are arising, which do not only affect the calculation of prices themselves, but also the broader economic influence on earnings of countries [12][13][14]. Therefore, the algorithm provided in this paper contributes both to cost calculation for taxation (and other business related cost and management problems) and to solve more general taxation questions.

The algorithm presented in this paper calculates work times for usage in cost calculation. From the five standard transfer pricing methods we chose Cost Plus for our implementation and demonstration as it is based on cost calculation [9]. Cost Plus is a method, where the transfer price is determined by calculating all the costs occurring in producing the transferred good or property. One commonly used way to calculate costs is the full absorption costing, which differentiates all costs of a business unit between costs directly assignable to a product or service (in our case direct to the transaction) or those who cannot be directly assigned to a product (indirect costs) [15]. For Cost Plus, the sum of direct and partial indirect cost is calculated, before a profit margin is added on top. Our approach helps to split the indirect costs of a business unit according to the real work effort for each product, as the problem statement will show more in detail (Section 4.1). Therefore, our solution targets calculating the work distribution between certain projects and not the time spent on each individual commit.

2.2. Inner source software development

Inner source is the use of open source best practices inside a company [2].

Like in open source, in inner source, developers lay open all project or product artifacts for everyone to see (only within the company, not publicly like in open source) [16][17]. They want other developers to find their code (open source typical self-selection) and start using it. The hope is that users identify bugs and help the code mature. Eventually, code contributions might flow back to the original developers (through peer-reviews), helping share development costs [3] [1].

As a literature review from Edison et al. [18] showed, inner source and its research is deeply integrated within a wide range of business aspects including knowledge & business management, business model design, and collaboration measurement. Being able to determine costs and software related intellectual property flow therefore is not only a matter of traditional, already well defined costing, accounting and management methods (e.g. [19] [20] [5] [8]), but is also important for inner source within industry [21].

Use of, contribution to, and collaboration on inner source projects is not apriori restricted. Thus it can happen across organizational boundaries, as developers begin to look past their silo. Such silo boundaries might be the boundaries of legal entities and hence any collaboration across these boundaries might be taxation relevant and hence require the calculation (and payment) of transfer prices.

3. Research Approach

We take a design science approach for our research. Design science is a methodological research framework that is used when researchers not only want to empirically analyze a situation, but also want to develop innovative solutions to real-world problems. Of several defined design science methodologies, we choose Peffers et al. [22], because of its ease of use.

Design science, according to Peffers et al. [22] is an iterative process, consisting of six main steps (plus an additional communication step, omitted here). The process centers on the creation and evaluation of an innovative artifact. This design science artifact should be a novel solution for an identified problem. The steps correspond with the following activities:

1. Problem identification. The researcher identifies and motivates a problem. In section 4, we present our past work and current literature review to identify the need for determining the (financial) costs of inner source collaboration.

2. Objective definition. The researcher defines the objectives for a solution to the identified problem. Our objectives (presented in section 4) are building on the problem identification and therefore are originating from the needs identified with our partners.

3. Solution design. The researcher develops a novel and innovative design science artifact according to the objectives. The solution we developed is based on statistical data gathered with our industry partner. The result is a method including an algorithm, presented in section 5.

4. Implementation. The researcher implements the solution for purposes of demonstration and evaluation. Implementation in our case indeed is the software implementation of the method and the algorithm, also presented in section 5.

5. Demonstration. The researcher first demonstrates the solution by applying the artifact to at least one instantiated problem. From past work, we use a real-world data set to demonstrate our solution, and present it in section 6.

6. Evaluation. The researcher evaluates the artifact using an appropriate method. We use member checking to gather feedback on our novel method and present the results in section 6.

The design science process allows for iteration, and this is also how we performed our work. In our case, we statistically analyzed gathered data, implemented prototypes and doing interim evaluations for ourselves in iterations. The presentation in the following sections is mostly linear, however, and focuses on the results.

4. Problem and Objective

The first and second step of the design science approach are the problem identification and objective definition, which will be explained in detail in the following two section.

4.1. Problem Identification

The problem identification was done in cooperation with our partners who later evaluated our approach with us. Additional literature research finalizes the problem identification. We saw, that inner source development can be taxation relevant (See 2.2). As inner source collaboration happens between business units, a need for calculating costs of transferred work between these units exist.

In the past it was sufficient to calculate costs of business entities coarse-grain, as no fine-grain differentiation between different projects of one business unit was needed. This changes with inner source, as contributions to software owned by different organizational and business units might be made. Moreover, the contributions are by the minute and not only once in retrospect for a larger period of time. Additionally, inner source contributions cannot be determined in advance (in cost and size). Three interviews conducted with industry partners through the data gathering process in previous work confirmed that these problems do occur in daily business. One interviewee was a software architect, one interviewee

was a (Scrum) product owner, and one other interviewee was an engineering manager. The interviews showed that calculating transfer prices is up to now solved by roughly estimating the costs rather than determining them more exactly through an algorithm. This leads to insecurities in terms of fiscal correctness and management acceptance.

In addition to calculating transfer prices, our solution might contribute to improving team management, utilizing inner source advantages (e.g. [23] [24] [1]) and handle organizational challenges coming with software development: For instance, functional organization is considered harmful for software engineering [25] and platform-based teams are still bringing many implementation barriers with it, as middle managers fear to not reach personal performance goals, if contributions to outside units are made [4]. Future research must show, how this paper can best be used to help introducing inner source within companies.

Finally, measuring working time for commits helps splitting indirect costs (e.g. personnel costs of developers) for usage within full absorption costing (see section 2.1) more exactly, as such a measurement is only coarse-grain up to now (see sections 1 and 2).

4.2. Objective Definition

Our literature review and industry interviews, as presented in the problem identification section, showed an unresolved mismatch between the need for correctly pricing software supply relationships and the demands of inner source software development and high-frequency code contributions across taxation boundaries. To that end, we define the objective for our research:

To develop a new implementation of the Cost Plus method that

- can correctly determine transfer prices for code contributions in client-supplier relationship, and
- can handle high frequency code collaboration between client and suppliers,

where ”high frequency” means many times daily as it is common in inner source software development collaboration. As we have the goal to develop an algorithm for cost calculation we want to find a way how to measure the distribution of the working time per project rather than measuring the exact time spent on each commit.

5. Design and Implementation

The third design science step is to present the solution design (section 5.1). After that (fourth step, section 5.2) the implementation will be explained.

5.1. Solution Design

Developing the solution design was done in several iterations by repeatedly analyzing commit data gathered in cooperation with our industry partners and improving the algorithm proposed in the previous iteration. The idea of the solution in this paper is to calculate working times from commits, which then might be used in different use cases. As this paper focuses on the use case of calculating transfer prices (using Cost Plus) for inner source development, the main result to be calculated is the cost share. Therefore, the basic solution design targets more an exact cost share for each organizational unit, then exact working times per commit. This also means, that organizational overhead will not be excluded in the calculation as the assumption is, that the overhead for each project is distributed equally to the work effort put into the project.

The concept was developed using commit data of a large multi-national corporation containing commits to a software platform from about 400 developers organized in 94 organizational units spreading over four hierarchy levels. The data-set was gathered continuously over one and a half years containing 230,000 file changes of 29,000 commits from 13 inner source projects/components. Due to the large amount of available data we assume, that the commit behavior (e. g. commit times, intervals between commits, LOC per commit) is representative for development work happening in companies. In future research we want to verify data and concept by conducting further studies in other multinational companies.

Required data. The following information are needed to design and implement our algorithm:

- Commit data (e.g. from Git): Author, Lines of Code (added, modified, deleted), timestamp, file path, commit identifier, project identifier
- Organizational hierarchy, incl. headcount
- Project list incl. owning organization
- Developer list and their organizational unit

Development process. We developed the concept by iteratively performing statistical analysis of the commit data and interim evaluations of implemented prototypes.

After all, we were able to identify two points of view: 1) Looking at the time difference between two

commits of the same author and 2) Looking at the commit timestamps. As first analysis we plotted the time differences of two commits from the same author. The results show, that most of the commits were made frequently within minutes or hours (mostly within 12 hours/720 minutes), but also significant number of commits are made once a day (culminating at about 1440 minutes/24hours). The same pattern can be observed for larger time differences (mostly commits with up to seven days in between). Moreover we found out (through analyzing the Lines of code committed and number of files changes) that committing at least once a day is the usual commit behaviour and that commits longer apart were e.g. mostly weekends or holidays. For the timestamp analysis, the commits were

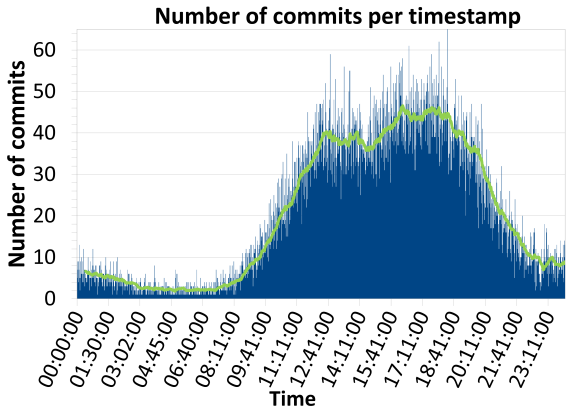

Figure 1. Number of commits per timestamp

grouped and counted by its commit time. Figure 1 shows the plot. Most commits are made during the day, but the sample data also contain commits made during the night. The data show, that developers are having flexible working times, which must be considered during further calculations.

Additionally performed analysis steps during will be described at the appropriate chapter.

Concept design. This chapter is about how working times are calculated. Cost calculation is done in the implementation step. The working time calculation concept is based on the time difference and timestamp distributions. Our previous results showed, that most commits are made at least daily. Resulting from that insight, the commits are mainly differentiated by time difference into two groups.

The first group are those, which are one working day (36 hours/2160 minutes, resulting from the previous analysis) or less in time difference. This time difference also includes commits being done in the evening with the previous commit being in the morning of the previous day. In short, all commits that belong to this group are those that regularly contribute to the software on a daily basis.

The second group of commits are all those, which are irregular. This is either the case if the time difference to the previous commit is longer than 36 hours apart or if the commit has no time difference at all (single commits by one author). Consequently, this group for example is suited for all first commits after weekends, holidays or long time without contribution, before beginning frequent development again.

Regular working time assignment. All regular commits (time difference <2160 minutes) are differentiated again by the time difference, as our previous analysis results have shown, that different use-cases are given, depending on the commits timestamp and the time difference. Additionally, it is more likely that commits with a small time difference are having a longer working times than those covering a night with larger time differences. In detail, three cases are differentiated:

1. <360 min. time difference
2. >=360 min. & <= 720 min. time difference
3. >720 min. & <= 2160 min. time difference

<360 minutes: The first group of commits (26% of all commits) are those with time difference less than 6 hours. These commits (mostly made within a matter of minutes) are getting working time equally to their time difference assigned, as there is a close timely relationship between the previous and the current commit.

>720 minutes & <= 2160 minutes time difference: The second group (mentioned third above, 22.7% of the commits) are all commits being 12 to 36 hours after the previous one. These commits are most likely to cover at least one night as they span a greater range of time as our analysis have confirmed. Consequently, these commit cannot have working time assigned in height of their time difference.

Commits belonging to this group are calculating working time proportional to the typical commit time distribution. Basis for the calculation is are the number of commits per timestamp (See Figure 1). The idea is to calculate the number of commits made betweeen the timestamps of the current and previous commit. The number of commits are summed up and set in proportion to the overall number of commits recognized. This ratio is then set into proportion to the length of a typical workday. Putting the algorithm into an equation, results in:

$$wt_{prop} = \frac{\sum_{i=timestamp_{cur}}^{timestamp_{prev}} \mathrm{n}_i}{\mathrm{n}_{overall}} * \mathrm{wt}_{day} \qquad (1)$$

wt represents the working time (of a working day day/ the result $prop$), calculated by iterating over the timestamp of the current (cur) until previous ($prev$) commit, with being n the number of commits.

The effect of applying the working time proportional to the number of commits is, that commits having a time difference of exact 24 hours, get a whole typical working day assigned. If the time difference is larger (e.g. 1 1/2 working days), the commit gets more working time. Additionally, this formula also minds, that some developers prefer working into the night by calculating the night time as working time as it is typical over all commits.

>=360 minutes & <= 720 minutes: The third group (mentioned secondly above, 4.7% of the commits) are all commits between 6 and 12 hours after the previous commit. This means, that there is not necessarily a night between these commits with developers either having a long work day or having a shorter night. The data set showed, that commits between both variants are not unusual and hence must be differentiated.

Commits belonging to the 6 to 12 hours group are treated differently depending on the share of the night they are covering. For this calculation, at first, it must be calculated how many minutes of the night are covered by the commit($nightshare = ns$):

$$ns = \frac{\sum_{i=timestamp_{cur}}^{timestamp_{prev}} x_i * 1}{\sum_{i=timestamp_{nightBegin}}^{timestamp_{nightEnd}} 1} \qquad x_i = \begin{cases} 1, & \text{if } i \in nighttime, \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

This formula counts for a commit, how many of the total minutes of a night (denominator) were covered (numerator). The result (a percentage) is used to calculate the working time:

$$wt_{ns} = ns * wt_{prop} + (1 - ns) * daycoverage$$

$$daycoverage = \sum_{i=timestamp_{cur}}^{timestamp_{prev}} x_i \qquad x_i = \begin{cases} 1, & \text{if } i \in daytime, \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The logic behind calculating with nightshare is, that a commit gets as much proportional working time (see Formula 1) as it covers the night (ns). This ensures, that commits covering large parts/the whole night get working time typical for the night assigned. The minutes belonging to the day ($daycoverage$, sum of minutes during the day), are fully assigned.

Example: The night is 10pm - 7am. Commit 1 is at 8am, the previous one at 9pm. As it covers the full night ($ns = 1$), it is fully proportional calculated. Commit 2 is at 4pm, the previous one at 6am. Commit 2 covers only 11% of the night ($ns = 0.11$), consequently this amount is calculated proportional. The minutes during the day (7 am to 4 pm) are calculated to 89%.
Comparing these two examples, we can see, that Commit 1 (done early in the morning) got less working time. Commit 2 (rarely goes into the night) got almost regular working time assigned.

For the calculation, we need to set two timestamps as the beginning and end of the night. For the implementation, this might be done by setting those as a fix value (depending on the dataset/ business entity) or by calculating a meaningful start of the day automatically. The latter was chosen in our implementation by calculating it dynamically through looking at the number of commits per timestamp (Figure 1) and setting the night begin/end to a certain percentage (15% in our case) of the highest number of commits.

Irregular working time assignment. All commits without time difference (1.4% of the cases) or not daily committed (45% of the cases) need a different handling as they don't have any (close) timely relationship to other commits, on which the calculation can be made. As part of the earlier described development process, additional analysis where done in later iterations to dive deeper into the commit behavior of developers.

Building on the previously described development steps, the relationship of working times, time differences and LOC of all regular commits were plotted. Figure 2 shows, that the number of LOC is exponentially rising, but the time differences and working times are linear. Consequently, we can conclude, that it is not possible, to assign working time just by looking at the LOC of a commit. For every LOC, a wide range of time differences & working times are possible, and for every time difference or working time, a wide range of LOC are possible. Thereafter, average and quartile working times for each LOC where plotted (Figure 3). This gives a deeper look into the working time distribution and LOC relationship. Through calculating linear regression lines, the overall trend is visible more easily, resulting into close median and average working times per LOC. Assuming, that the data representative, the median regression line can now be taken to estimate the working time for all irregular commits. Using this method, the irregular commits also get working time

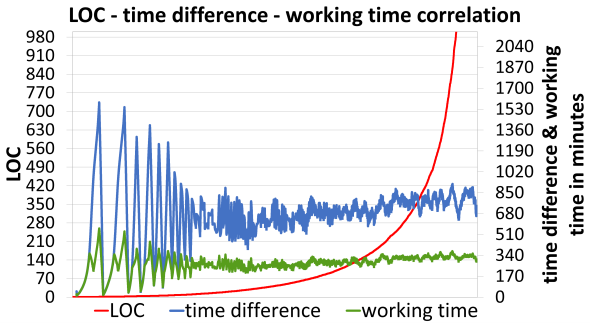

Figure 2. LOC, time difference and working time correlation

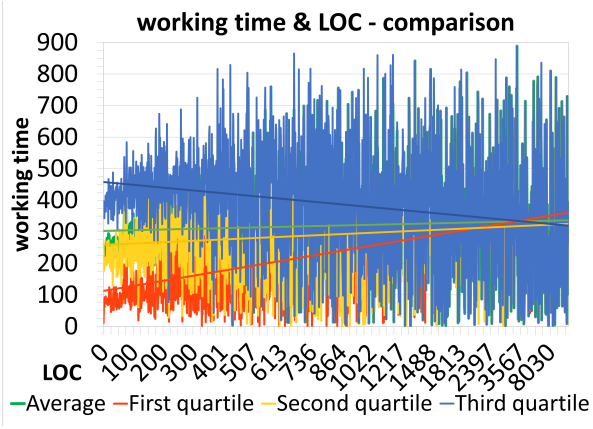

Figure 3. LOC and working time comparison

assigned based on the results of the regular commits:

$$wt_{linear} = 0.04267525 * loc + 258.58249058 \quad (4)$$

5.2. Implementation

For the implementation, the commit data (available in CSV format) where parsed and uploaded to a PostgreSQL database alongside all other information needed (See section 5.1). To make the data easier reusable in the future, a REST API was developed in PHP, returning the transfer price in JSON format, running on an Apache Webserver. The steps not only represent the processing order, but also the development process of the implementation.

The **first step** in the process was to upload the CSV commit data into the database. In this step, preprocessing was done to eliminate invalid data.

The **second step** was to receive cost data for each business entity (cost centre view). In our sample implementation no real-life data but dummy costs were used to show the basic cost calculation procedure.

The **third step** was to load the commit data from the database, process them (e.g. calculate time difference) and calculate the algorithm presented in the solution design section (Section 5.1). The result after this step is a list of commits, on which every commit has its working time assigned according to our solution design.

As already said, for our sample use case, share of the workload between all contributed projects of entity is important. Therefore, the **fourth step** was to aggregate the list of working times to the needed organizational level. We aggregated our sample data on a team-level, as this hierarchy level is the lowest available cost centre level. For each entity, the sum of working times per projects are calculated. Out of the sum of working times, the work share distribution can be calculated (e.g. 30% to Project A, 60% to Project B, 10% to Project C).

The **fifth step** was the calculation of the transfer prices, in our example using the Cost Plus method. This means, that for each entity, a value must be calculated which flows to other entities. The result of step four (work share distribution) is used for the calculation. As we also have the costs per business entity, we can now split all costs which are not directly assignable to one transaction (indirect costs e.g. developers salary) according to the work share spend on it. On top of all the costs, a profit margin is added (5% flat in our case as an demonstration example).

The end result is a list of organizational units and their costs per project they are developing. These data are returned in the **sixth step** over the REST API in JSON format and may be used in various use cases like controlling or taxation.

6. Demonstration and Evaluation

The last two steps in design science are the solution demonstration and evaluation.

6.1. Demonstration

We demonstrate our solution using the real world data on which the algorithm is based on. Before the cost calculation itself will be demonstrated, the work time assignment for each of the four cases (3 regular commits, 1 irregular without time difference) will be shown. The first example shows part of a JSON output from the API belonging to **Case 1** (time difference <360 min.):

```
{"module": "Module A",
"commit_date": "2015-01-02 15:18:02",
"owner": "Person A",
"time_difference": "23",
"loc": "462",
"working_time": "23"}
```

As the commit was done close after the previous

commit, it gets its time difference as work time assigned. The implemented API returns not only the working time of the commit, but also additional data (e.g. time difference, name of committer, the module and LOC). Additionally, organizational data for the later cost calculation are included (not shown here).

Case 2 (360 - 720 min. time difference) can be shown with a commit, which was done in the morning after covering large parts of the night (previous commit was at 10:40 pm). Consequently, the commit does get less work time assigned for the night time. In our demonstration we can see, that the algorithm works exactly like desired: Commits covering the night get not the full night as work time:


```
{"module": "Module B",
"commit_date": "2016-02-15 09:35:27",
"owner": "Person B",
"time_difference": "655",
"loc": "70",
"night_share": 1,
"working_time": 50}
```


In this example, the commit covers 655 minutes, mostly over night. Due to the calculations of our algorithm, it gets 50 minutes of work time assigned, which means the developer effectively started programming (according to the statistical correlation to other commits) at 8:45 AM.

Case 3 (720-2160 min. time difference) is shown on a commit covering part of two work days (covering 16 hours through the night). Therefore, work time proportional to the statistical typical commits covering that time is assigned:


```
{"module": "Module C",
"commit_date": "2016-04-08 11:07:01",
"owner": "Person C",
"time_difference": "953",
"loc": "684",
"working_time": 154}
```


In this example, the previous commit was done at 7:14 PM and therefore covers not only the night, but also parts of the current work day. Our algorithm assigns the commit working time of 154 minutes, which (if the developer stopped working at 7:14 PM) effectively means the committer started working on the module at 8:33 AM.

Case 4 (Irregular commit) shows, that commits with no or to large time difference get reasonable work time based on the other commits by using the median regression line from the Solution section:


```
{"module": "Module D",
"commit_date": "2015-01-06 17:58:19",
"owner": "Person D",
"time_difference": null,
"loc": "227",
"working_time": 268}
```


In this case, the commit to module D was made by Person D, which only made a single commit to the software platform. The commit got work time (268 minutes) assigned in height of the median value of all regular commits (demonstrated with case 1 to 3) with equal LOC.

All working times are aggregated for each cost centre. In our second demonstration, the lowest organizational level with dedicated cost centres was taken. All commits belonging to this organization (committer is located there) are aggregated according to the target organization (owner of the module). Based on the total work times per organization, a working time share is calculated, which represents the amount of work every module the organization worked on. The percentage share is then used for splitting the indirect costs in that respective ration. After adding the profit margin (5% flat in our case), the transfer price can be calculated:


```
{"org_name": "Organisation-A-A-A",
"parent_org": "Organisation-A-A",
"transfer_data": {
"2015": {
"Organisation-A-A": {
"share": 84.96,
"costs": 47701.28,
"transfer_price": 50086.34
},
"Organisation-A-A-A": {
"share": 7.31,
"costs": 4106.04,
"transfer_price": 4311.34
},
"Organisation-B-A": {
"share": 6.53,
"costs": 3669.06,
"transfer_price": 3852.51
}[...]}}}}
```


This example transfer prices for the rather small sample entity show that in 2015 most of the commits (92.27%) are still within the same organization (*Organisation-A-A-A*) or for the parent business unit (*Organisation-A-A*). A smaller part (in sum 7.73%) is crossing tax boundaries to other business units (*B-A, etc.*) and therefore might be taxation and accounting relevant.

Commits to projects, which were labelled by the company as not relevant to the analysis where excluded in the analysis and consequently also not included in the transfer price calculation.

6.2. Evaluation

We evaluated our approach using member checking for our demonstration data and feedback from experts on the subject matter. In total, we conducted interviews with six people having three different views on transfer pricing in inner source: Two people form the German Ministry of Finance, one person from a large international account firm and three people from our industry partner (previously involved in problem identification). Our goal was to evaluate two main aspects: 1) Evaluated the basic usage of Cost Plus for transfer pricing within inner source. 2) Evaluate the algorithmic suitability of the algorithm itself.

Throughout our interviews and further collaboration with the German Ministry of Finance, we were able to analyze the different possible transfer pricing methods and its prerequisites to be met when applied to inner source development. A result is that Cost Plus is applicable (and first choice) for inner source development, when a client-supplier relationship is given [26]. Our interview with the large international accounting firm confirmed that, even though they emphasized, that choosing a method strongly depends on the individual case. We consider our evaluation objectives to be fulfilled, as we specifically targeted Cost Plus.

Our second main evaluation goal (suitability of the algorithm for calculating cost plus) was also evaluated with our interview partners at the German Ministry of Finance. The head of education for the department confirmed that our solution is not only a valid way of calculating Cost Plus for transfer pricing, but also offers a significant advantage compared to the way of determining the Cost Plus transfer price up to now. This confirmed the usability of our algorithm from a tax officials point of view.

Evaluating the algorithmic suitability from an industry point of view was done again with the three industry interview partners already involed in data gathering and problem identification (member checking, details to interviewed persons see section 4.1). As original data source they knew the data, its structure, and its corporate context. They confirmed that our results fit the real world. Additionally, they attest to us that the working time share was useful and fits the internal work flows. They challenged the algorithms ability to correctly calculate the time spent on an individual commit. However, in the context of calculating the Cost Plus method, we only care about the working time distribution (percentage-shares) and not individual commits. Hence, their general confirmation of the usefulness of our approach stands.

7. Discussion and Limitations

We present a novel algorithm and its implementation for more precise Cost Plus calculation. The method is defined by the OECD and is commonly used in calculating transfer prices in client supplier relationships. We move past current practice, which takes a coarse-grain approach, by basing our algorithm on the actual code contributions. Using our algorithm, we can calculate the time and associated labor costs to determine transfer prices between a supplier and their client. The demonstration and evaluation show that we succeeded on both our defined objectives: A new and correct algorithm that performs well when faced with a large amount of individual (but high-frequency) code contributions.

Additionally, our demonstration and evaluation also scopes this research: We do not move into acceptance or analyzing wide-spread use of our work (yet). Hence, empirical evaluation criteria like trustworthiness for qualitative research [27] or reliability and validity for quantitative research do not apply. We therefore limit our discussion to general aspects of our work.

The most obvious limitation is that on the one hand, we aim to provide a new algorithm for precisely calculating labor costs in software development. Yet, towards this general goal, we are missing the final step, which is to determine the error of the time spent on an individual code contribution. This error is most likely a distribution around the value we calculate, but we have yet to determine this. For the specific purposes of our research, as set out in the objective section, this missing piece does not matter: For calculating Cost Plus precisely, the percentage relationships are sufficient so that we don't need the absolute values.

In terms of computational efficiency, the algorithm scales linearly with the number of transactions (commits). Given that it is our goal to be precise and fine-grain, we have to go down to the level of accessing each individual commit, and hence a linear performance is the best we can get.

8. Conclusions

As already shown in the discussion and limitations section, we developed a new algorithm to determine working times spend on commits for usage within cost calculation, especially Cost Plus in transfer pricing.

Even though, the algorithm and its implementation is limited to calculate work distribution for organizations, it sets the basis to enable exact work time calculation per commit. Therefore, future research can focus on improving the accuracy of individual commits, which

enables more application fields for the algorithm.

Moreover, future research needs to show which additional areas can benefit from our algorithm, what adjustments need to be made. From an inner source perspective it is worth finding out, how the results of this paper can best be used to improve acceptance of inner source and bring economic advantages with it.

9. Acknowledgment

We would like to acknowledge Elçin Yenişen Yavuz, Thomas Wolter and the anonymous reviewers for their valuable feedback that helped us improve the paper significantly.

References

[1] M. Capraro and D. Riehle, "Inner source definition, benefits, and challenges," *ACM Comput. Surv.*, vol. 49, Dec. 2016.

[2] K.-J. Stol and B. Fitzgerald, "Inner source–adopting open source development practices in organizations: A tutorial," *IEEE Software*, vol. 32, no. 4, pp. 60–67, 2015.

[3] M. Capraro, *Measuring Inner Source Collaboration*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2020.

[4] D. Riehle, M. Capraro, D. Kips, and L. Horn, "Inner source in platform-based product engineering," *IEEE Transactions on Software Engineering*, vol. 42, pp. 1162–1177, 12 2016.

[5] The International Federation of Accountants, *Evaluating and Improving Costing in Organizations*. 2009.

[6] C. Ebert, "Software product management," *IEEE Software*, vol. 31, no. 3, pp. 21–24, 2014.

[7] C. Ebert, B. K. Murthy, and N. N. Jha, "Managing risks in global software engineering: Principles and practices," in *2008 IEEE International Conference on Global Software Engineering*, pp. 131–140, 2008.

[8] T.-H. Cheng, S. Jansen, and M. Remmers, "Controlling and monitoring agile software development in three dutch product software companies," in *2009 ICSE Workshop on Software Development Governance*, pp. 29–35, 2009.

[9] OECD, *OECD Transfer Pricing Guidelines for Multinational Enterprises and Tax Administrations 2017*. 2017.

[10] United Nations, *United Nations Practical Manual on Transfer Pricing for Developing Countries*. United Nations, 2014.

[11] Y. Holtzman and P. Nagel, "An introduction to transfer pricing," *The Journal of Management Development*, vol. 33, 02 2014.

[12] OECD, *Aligning Transfer Pricing Outcomes with Value Creation, Actions 8-10 - 2015 Final Reports*. 2015.

[13] O. Mazur, "Transfer pricing challenges in the cloud," *Boston College Law Review*, vol. 57, no. 2, pp. 643–693, 2016.

[14] M. Olbert and C. Spengel, "International taxation in the digital economy : challenge accepted?," *World Tax Journal : WTJ*, vol. 9, no. 1, pp. 3–46, 2017.

[15] B. B. C. Aurora, "The Cost Of Production Under Direct Costing And Absorption Costing – A Comparative Approach," *Annals - Economy Series*, vol. 2, pp. 123–129, April 2013.

[16] Open Source Initiative, "The open source definition," 2007.

[17] K.-J. Stol, P. Avgeriou, M. A. Babar, Y. Lucas, and B. Fitzgerald, "Key factors for adopting inner source," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, Apr. 2014.

[18] H. Edison, N. Carroll, L. Morgan, and K. Conboy, "Inner source software development: Current thinking and an agenda for future research," *Journal of Systems and Software*, vol. 163, p. 110520, 05 2020.

[19] B. B. C. Aurora, "The cost of production under direct costing and absorption costing – a comparative approach," *Annals - Economy Series*, vol. 2, pp. 123–129, 2013.

[20] M. Kornberger, D. Pflueger, and J. Mouritsen, "Evaluative infrastructures: Accounting for platform organization," *Accounting, Organizations and Society*, vol. 60, pp. 79–95, 2017.

[21] G. Gruetter, D. Fregonese, and J. Zink, "Living in a biosphere at robert bosch," in *Adopting InnerSource: Principles and Case Studies* (D. Cooper and K.-J. Stol, eds.), 06 2018.

[22] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, pp. 45–77, 01 2007.

[23] N. Carroll, L. Morgan, and K. Conboy, "Examining the impact of adopting inner source software practices," in *Proceedings of the 14th International Symposium on Open Collaboration*, OpenSym '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[24] D. Cooper and K.-J. Stol, *Adopting InnerSource: Principles and Case Studies*. 06 2018.

[25] R. Fuller, "Functional organization of software groups considered harmful," in *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pp. 120–124, 2019.

[26] A. Neumann, "Transfer pricing in inner source software development," Master's thesis, Hochschule des Bundes für öffentliche Verwaltung, Bruhl, Germany, 2019.

[27] E. G. Guba, "Criteria for assessing the trustworthiness of naturalistic inquiries," *ECTJ*, vol. 29, p. 75, Jun 1981.