Johanna Kaihlavirta

Time tracking in software maintenance service

Master's Thesis in Mathematical Information Technology

November 8, 2022

University of Jyväskylä

Faculty of Information Technology

Author: Johanna Kaihlavirta

Contact information: kjpelkonen@gmail.com

Supervisor: Antti-Jussi Lakanen and Ville Seppänen

Title: Time tracking in software maintenance service

Työn nimi: Tuntiseuranta ohjelmistoylläpitopalvelussa

Project: Master's Thesis

Study line: Specialisation in Software and Telecommunication Technology

Page count: 60+2

Abstract: Software maintenance takes a major part of the development life cycle for a system in both time and money. Developers in maintenance teams are working with multiple tasks parallel and interleaved. They report effort in timesheets which are then used for customer invoicing and estimating future work - both which are important to get right. However, the accuracy of time reports may vary. Literature identifies several reasons for inaccurate reports but it is not researched how these reasons are solved in practice. Literature suggests a way for setting up a time tracking system and use cases for gathered data, but doesn't examine the actual use of a time tracking system, nor how companies can instruct developers to report their time accurately.

This study proposes instructions aiding in task categorisation for time reporting in software maintenance context. Design science research method was applied in a consultancy company and its software maintenance team to make the time tracking process easier. The team took part in defining activities and mapping them to current categorisation. The defined activities were used to construct an artifact of instructions in the form of a decision tree. The tree provided sixty examples for task categorisation and it was evaluated in practical use by interviewing team members.

This study concludes that the artifact was helpful for learning task categorisation rules initially. The study also confirmed claims from literature regarding obstacles in the time track-

ing process, namely, perceived stress in justifying time spent on internal work or when exceeding an estimated time frame for customer work items, and recalling daily activities and time spent on each activity afterwards. However, the artifact designed in this study could not answer these concerns. The main finding is that time tracking was perceived as the most annoying part of the job. This observation reveals practical problems which need attention in further research.

Keywords: time tracking, time reporting, software maintenance, timesheet, billables

Suomenkielinen tiivistelmä: Ohjelmistojen ylläpitovaihe vie suuren osan ohjelmiston koko elinkaaren ajasta ja rahoituksesta. Ylläpidossa kehittäjät työstävät yhtä aikaa monta rinnakkaista tehtävää. Käyttämänsä ajan he raportoivat tuntiraportille, jota käytetään asiakkaan laskutukseen ja tulevan työn työmääräarviointiin - molemmat hyvin tärkeitä asioita tehdä oikein. Tuntiraporttien tarkkuus voi kuitenkin vaihdella. Kirjallisuudessa tunnistetaan vaihtelulle useita syitä, mutta syiden ratkaisemista ei ole tutkittu. Tuntiraportointijärjestelmän luomiseen ja tuntiraporteilta saatavan tiedon käyttämiseen on kirjallisuudessa ehdotuksia, mutta itse tuntiraportointijärjestelmän käyttöä tarkan tiedon saamiseksi ei ole tarkasteltu.

Tämä tutkimus ehdottaa ohjeistusta työtehtävien luokitteluun tuntiraportointia varten ohjelmistoylläpitotyössä. Tutkimus toteutettiin suunnittelutiedemenetelmällä konsultointiyrityksen ohjelmistoylläpitotiimissä. Tavoitteena oli helpottaa tuntiraportointiprosessia. Tiimi osallistui tehtäviensä kirjaamiseen. Kirjatut tehtävät muokattiin ohjeeksi päätöspuun muotoon. Ohje on tämän tutkimuksen ohessa tuotettu artefakti. Ohje auttoi tehtävien luokittelussa kuudenkymmenen esimerkin kautta, ja sen käyttöä arvioitiin tiimiläisten haastattelujen avulla.

Tutkimuksen lopputulokseksi saatiin vahvistettua artefaktin hyödyllisyys tehtävien luokittelun opettelussa. Lisäksi tutkimus vahvistaa kirjallisuudesta poimitut havainnot tuntikirjauksen hankaluuksista: koettu ahdistus sisäisen työn oikeutuksesta tai etukäteen arvioidun aikaraamin ylittämisestä asiakastyössä, sekä päivän aikana tehtyjen työtehtävien ja niiden keston muisteleminen jälkikäteen. Aftifakti ei pystynyt ohjeistamaan työntekijöitä näiden huolten osalta. Päälöydös on, että tuntiraportointi koettiin työn ärsyttävimmäksi osaksi. Tämä havainto paljastaa käytännön ongelmia, jotka vaativat selvitäkseen jatkotutkimuksia.

Avainsanat: tuntikirjaus, tuntiraportointi, ohjelmistoylläpito

List of Figures

List of Tables

Contents

1 Introduction

Time reports are a way for software maintenance providers to make effort visible to customers (April 2010; Yakura 2001). However, time tracking is not a simple task (Sindhgatta et al. 2010). Turning effort into a time report, or a timesheet, is not straightforward. One developer can interpret an activity in several different ways for a timesheet, and two consultants are likely to interpret the same activity differently in their respective timesheets (Yakura 2001).

If effort is difficult to track for one project with only one customer, it is even harder in software maintenance, where it is custom to work for several customers and projects simultaneously (Sindhgatta et al. 2010). Several reasons for incorrect timesheets are recognised by Yakura (2001) and Sindhgatta et al. (2010)but literature does not provide much guidance to solve those issues.

Two case studies address a topic of creating a time tracking system. Shoukourian and Danielyan (2011) present how to set up a time tracking system for a maintenance team and how to prepare a team to use it. April (2010) explores maintenance task categorisation for time reporting purposes to enable spotting trends to guide maintenance activities and manage customer expectations.

However, neither of these studies address the use of a time tracking system. It is seen by April (2010) as a managerial task to ensure that employees track time accurately, and they give an example instructions. Their study does not examine reasons nor solutions for the problem of inaccurate time tracking. Unfortunately, research among software maintenance overall is scarce (Sharon Christa et al. 2017), and thus understanding the reasons for the complexity of time tracking remains mostly unknown. Investigating human and organisational factors in software maintenance is in-depth covered by Bhatt, Shroff, and Misra (2004) and Bhatt, Shroff, Anantaram, et al. (2006), but the research lacks the connection to time tracking.

Insight on what makes time tracking difficult and inaccurate could aid improving time tracking processes and making it less complex. Time tracking is a never-ending activity, and less expensive if it is easy to do (Shoukourian and Danielyan 2011). Inaccurate timesheet data

can result in under- or over-invoicing customers, and creates a wrong baseline for future estimations (Sindhgatta et al. 2010; Yakura 2001).

The research question for this study is: *What kind of instructions would aid in the time tracking process for software maintenance service?*

Possible solutions to answer this question of instructions were sought from literature, and two solutions were derived from case studies of April (2010) and Shoukourian and Danielyan (2011). Design science research method was used to implement solutions in a market leading, Finland based IT consulting company in the field of cloud technologies. A software maintenance team from the company acted as a target group. The author was a supervisor in the team and collaborated closely with developers within this research context. With the results of this thesis, the author hoped to ease the burden of her own team regarding time tracking practices.

The first step towards instructions was to conduct a mapping exercise with the team. It was inspired by Shoukourian and Danielyan (2011) to include the team to map activities to the current time tracking categorisation. Data gathered from this exercise was used to create detailed instructions in the form of a decision tree, as was done in the case study of April (2010).

Categorisation instructions is the artifact which this research produced. It was presented as a PDF file and freely available for the team. The tree asked what activity a user had had, and answered which of the categories in the time tracking system was equivalent for the activity. It was notably detailed: it listed sixty activities. The tree was divided into sections of related activities to ease navigation. The tree also aided in building a metal model of categorisation rules by questioning a trigger for certain activities, and guiding forward to an equivalent category only after the user specified the trigger.

After the decision tree had been available for two months, eight employees from the team were interviewed. They were asked about how maintenance work appeared to them, how they tracked and reported their time, how the mapping exercise and decision tree had helped them, and how they felt about time tracking in general.

The answers revealed that the detailed instructions were mainly useful for employees with short tenure. They hadn't yet built a mental model of time tracking categorisation, so instructions with lots of examples were aiding them considerably. However, employees with longer tenure did not benefit from the instructions, because it did not bring new insight for them. Instead of having a problem of categorisation, employees with longer tenure faced trouble of justifying the used hours on an activity and recalling daily activities when it was time to write time reports.

Time tracking was perceived as the most annoying part of the job in the team. It was apparent from interviews that even with a complete mental model of time tracking, employees felt stress about time reporting and experienced time tracking to be cumbersome. The author feels the most important result from the research was to understand different problems for time tracking and reasons for stress. Stress was caused by exceeding estimations, pressure from management to include administrative work within customer work, and simply recalling what tasks an employee had had during a day and how much time was spent on each task.

We can say that the research provides one solution for one use case in time tracking context: detailed instructions made time tracking easier for new employees to learn task categorisation rules. The research provided more information in the problem domain and made possible for designing new solution candidates to aid in other aspects of time tracking than learning the task categorisation.

This thesis is structured as follows: in Chapter 2, we investigate literature to define software maintenance, its activities and benefits of outsourcing. In Chapter 3, we examine literature to define the purpose of time tracking and obstacles that can be faced while reporting time. The research design is described in Chapter 4, and the resulting artifact is presented in Chapter 5. Research results are presented in Chapter 6, and the meaning of the results in the light of literature and practice is described in Chapter 7. Chapter 8 concludes this study.

2 Software maintenance

2.1 Software maintenance life cycle

Software maintenance is the activity to keep a software system alive and meet requirements dictated by a customer or environment (Sharon Christa et al. 2017). It is one aspect in the software life cycle, sustaining the software throughout its life. Its objective is to modify existing software while preserving its integrity. (Bourque, Fairley, and IEEE 2014, p. 5-2; IEEE 2022, p. 20–21).

The main purpose for software maintenance is to sustain the system's capability to provide a service (IEEE 2017, p. 103–104), to ensure that the software continues to satisfy user requirements (Bourque, Fairley, and IEEE 2014, p. 5-3), and to extend the life of a software system for as long as possible (Bourque, Fairley, and IEEE 2014, p. 5-5). The reliability and quality of the system is to be achieved with minimum effort, cost and time (Sharon Christa et al. 2017, p. 763).

Typically the maintenance phase in software life cycle lasts for many years. Software maintenance doesn't only happen after a software system is published to production. Planning for maintenance should begin when the decision to develop a new software system is made. (Bourque, Fairley, and IEEE 2014, p. 5-9). Development phase is the right place to pay attention to maintainability aspects, for instance, documentation and test environments. This will reduce maintenance costs after the development phase. (Bourque, Fairley, and IEEE 2014, p. 5-5).

Maintenance phase takes considerable amount of effort during the whole software life cycle (Bhatt, Shroff, and Misra 2004), and a notable amount of budget compared to new software development (IEEE 2022, p. vi) - over five times more than the development process (Sharon Christa et al. 2017), or 60% of the whole IT budget for a company (Rahman et al. 2020). However, software maintenance is not linear effort throughout time: on the contrary, it has peaks due to external factors, and finally a steady state with lowered effort per day (Bhatt, Shroff, and Misra 2004).

Maintenance effort consists of many different activities, caused by many different sources. We will investigate this in the next chapter.

2.2 Maintenance activities

Software maintenance includes all the activities and processes, and management of these processes, which modify existing software (Chapin et al. 2001). Broadly, software maintenance includes error corrections, changes and improvements to operational software (Bhatt, Shroff, Anantaram, et al. 2006), although software doesn't have to be deployed or delivered to production to qualify as the target for maintenance activities (Chapin et al. 2001).

The need for maintenance activities occurs from changes in the operational environment. Changes can occur to e.g. interfaced systems or infrastructure. Evolving security threats might be noticed or system elements might become technically obsolete. Need for correction might arise from a detected error, or new or modified capability is required. (IEEE 2022, p. 20–21; 2017, p. 103–104; Bhatt, Shroff, and Misra 2004). As the world is changing, so are requirements and enhancements for the software evolving (Sharon Christa et al. 2017), and it might cause the original requirements to be no longer applicable. (IEEE 2022, p. 20–21)

Maintenance is performed as a continuing series of prioritised work items (IEEE 2017, p. 103–104). It includes preparations, performing maintenance and logistics support, and managing results of these activities (IEEE 2022, p. 6). Activities in the maintenance process include monitoring the system's capability to deliver services, recording incidents for analysis, modifying code, records and information items, and confirming restored capability. (IEEE 2017, p. 103–104; 2022, p. 20–21).

Software maintenance is an industry separate from new software development (Bhatt, Shroff, and Misra 2004), albeit it has many activities in common with software development in addition to several unique activities (Bourque, Fairley, and IEEE 2014, p. 5-6). Shared activities include analysis, design, coding, testing and documentation (Bourque, Fairley, and IEEE 2014, p. 5-8). Usually, maintenance activities start with a change request, created from an issue raised by a customer (Sharon Christa et al. 2017). The relevance is verified and then the change request is passed for the maintenance team (Sharon Christa et al. 2017) or rejected

and redirected to a development team (Bourque, Fairley, and IEEE 2014, p. 5-8) - although it is recommended to include new development activities in the maintenance contract (IEEE 2022).

For the change request, the maintenance team makes an impact analysis, fixes an issue or estimates an enhancement (Sharon Christa et al. 2017; Bourque, Fairley, and IEEE 2014, p. 5-8). Estimation is then forwarded to a customer who either approves or disapproves the modification effort. Required resources are allocated, and finally, a report is generated. (Sharon Christa et al. 2017).

Prior to working with a software system, a maintenance team first needs to gain understanding in the software. This is usually done in a knowledge sharing activity during transition from the development team to the maintenance team. The maintenance team might also work in help desk activities, and it must comply with service level agreements (SLA's) according to a contract with a customer. (Bourque, Fairley, and IEEE 2014, p. 5-8).

To monitor and act on changes, companies use classification (Chapin et al. 2001). We will examine different classifications in the next chapter.

2.3 Categorisation of activities

During the maintenance process, all proposed changes are recorded as modification requests. A modification can be categorised as a correction or enhancement, which are further defined in the list below. (IEEE 2022, p. 4). These types were originally developed by Lientz and Swanson (1980), and later they became part of IEEE standard, although slightly inconsistently compared to the original definition (Chapin et al. 2001).

Current definitions from IEEE (2022) are

- adaptive maintenance: modification to keep a software system usable in a changed or changing environment
- corrective maintenance: modification to correct discovered problems
- perfective maintenance: modification to improve software, including features or information for users, and enhanced maintainability, performance, or other software at-

tribute

- preventive maintenance: modification to correct latent faults before they occur in the live system

According to Chapin et al. (2001), organisations use more detailed categorisation compared to the four broad types of maintenance. This allows companies to report, budget, staff and monitor activities in more detail. Chapin et al. (2001) defined twelve types of software maintenance, depending on the required changes. The types are defined in Figure 1. The figure shows how changes in source code functions might be caused by a need from technical software properties or environment, or a need from changed business rules. If software wasn't changed, a different set of types is defined for documentation changes, and the final set of types is defined for training, consulting or evaluating software. (Chapin et al. 2001).

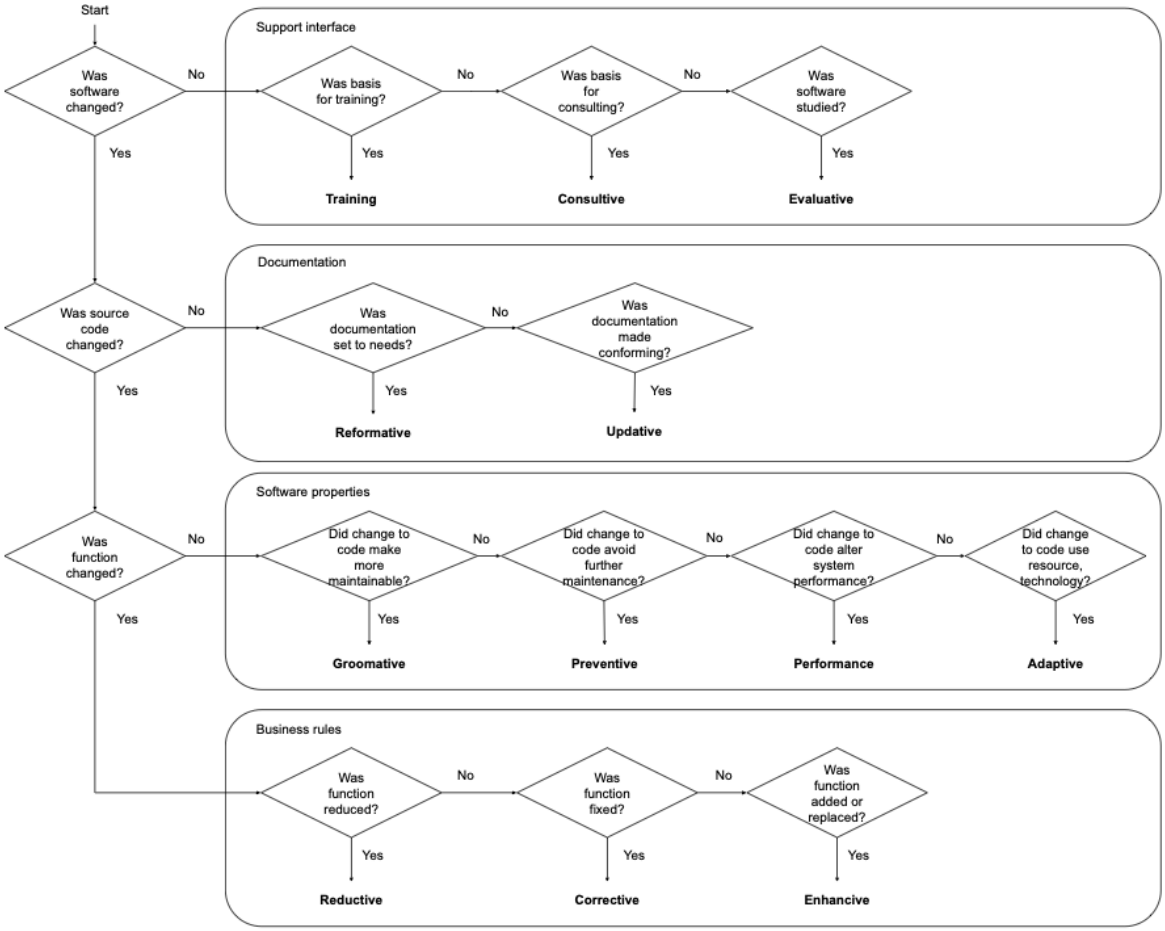

Figure 1. Types of software maintenance activities, adapted from Chapin et al. (2001, p. 10).

As we have seen, the maintenance process contains many activities. It all begins after the operation or development process, and it can transition between either of these and the maintenance process during the life cycle, ending in either disposal or transition process (IEEE 2022, p. 6).

Maintenance service can be produced by the software developing organisation, or outsourced to a third party. However, the same maintenance tasks prevail. (Sharon Christa et al. 2017). In the next chapter we examine the benefits of outsourcing maintenance to a permanent maintenance team.

2.4 Outsourcing software maintenance service

Software maintenance is not necessarily carried out by the team that developed the software (Bourque, Fairley, and IEEE 2014, p. 5-6). Actually, Bhatt, Shroff, and Misra (2004) observed that in every occasion of their target groups the maintenance team had nothing to do with the design and development of the software.

Software maintenance suffers a bad reputation among software development life cycles. It is a neglected part of software development in both the amount of research, and respect the work enjoys among software developers. (Sharon Christa et al. 2017; Bourque, Fairley, and IEEE 2014, p. 5-1; Bhatt, Shroff, and Misra 2004). Maintenance work is viewed by developers as a less creative and high workload job (Sharon Christa et al. 2017; Bhatt, Shroff, and Misra 2004). Thus, rotation of staff is frequent. (Sharon Christa et al. 2017).

However, a permanent maintenance team has several benefits over combined development and maintenance teams. These benefits are, for instance, opportunities to specialise, established communication channels, collegiate atmosphere, reduced dependency on individuals and possibility for audit checks (Bourque, Fairley, and IEEE 2014, p. 5-6). Luckily, the perception of software maintenance is now changing as companies focus on keeping software operating as long as possible to get the most out of their investment in development. Also, the open source paradigm has increased attention in maintaining software developed by others. (Bourque, Fairley, and IEEE 2014, p. 5-1).

Outsourcing software maintenance has become a major industry. A large initial investment is needed in determining the scope of maintenance and contractual details, including the terms for service level agreements (SLAs). (Bourque, Fairley, and IEEE 2014, p. 5-6). By offshore outsourcing, companies are looking forward to reducing overall maintenance costs (Rahman et al. 2020). The organisation owning a software system and the company providing maintenance services should agree upon maintenance service models. The types of maintenance should be addressed, and new development included in the agreement. The agreement can be either with a fixed price or with a time and materials (T&M) cost basis. (IEEE 2022, p. 21).

On the T&M model the services are paid on the basis of effort or the number of engineers engaged in maintenance. This was more common earlier, and now there's a shift towards SLA-based fixed-price services over a specified period. (Bhatt, K, et al. 2006). Many different combinations of T&M and fixed price are possible, with extremes of a blanket contract with fixed price for both maintenance and new development, and a specific contract for only corrective maintenance without new development. (IEEE 2022, p. 21).

Rahman et al. (2020) has proposed ten critical factors for a successful software maintenance outsourcing decision. The factors concern the customer, the provider, code base and processes. These factors were later verified by Rahman et al. (2021). We will investigate some of these factors and combine them with identified quality attributes of successful maintenance teams by Bhatt, K, et al. (2006) and Sharon Christa et al. (2017) in the next chapter.

2.5 Factors for successful service

The skillset of the maintenance team plays a critical role in providing a maintenance service successfully (Sharon Christa et al. 2017) and is one of the critical factors for an outsourcing decision (Rahman et al. 2020). Investment in training employees indicates higher organisational commitment, and along with good leadership leads to a more qualified and stable maintenance team. Stronger leadership also increases training of end users, whereas understaffing the maintenance team decreases end user training. (Bhatt, K, et al. 2006).

Available documentation and knowledge transfer from the initial development team to the

maintenance team is important for a successful service (Sharon Christa et al. 2017). Knowledge transfer can be arranged by involving initial developers in the maintenance team (Bhatt, K, et al. 2006) or involving maintenance team members early in the development phase (Bourque, Fairley, and IEEE 2014). Facilities, e.g. audio/video conference between development and maintenance team, are found to support knowledge transfer (Bhatt, K, et al. 2006).

Communication is seen as cumbersome between separately located teams of a customer and a software maintenance provider (Rahman et al. 2020). However, it's a critical factor to succeeding in outsourcing (Rahman et al. 2020) because the key to effective maintenance is communication of the requirements from a customer to the maintenance team in a clear, consistent and complete manner (Sharon Christa et al. 2017). Close partnership enables smooth knowledge transfer, flow of information, and understanding the business requirements. Communication takes a lot of time for software engineers, so proper management of knowledge sharing and transfer is important for both parties. (Rahman et al. 2020).

Good cooperation between the maintenance team and the customer increases the maintenance team's capability via smoothness of knowledge transfer (Bhatt, K, et al. 2006). It's important for the customer to have IT knowledge to work smoothly with maintenance service providers (Rahman et al. 2020). Even the size of the maintenance team can be smaller if the customer has an experienced IT team (Bhatt, K, et al. 2006).

The amount of end users, system complexity and strict service level agreements are proven to affect maintenance effort. Lower tolerance for slippage increases the needed total size of the maintenance team. These factors also affect organisational climate by the difficulty of targets set for the maintenance team. Organisational climate can be observed in the form of pressure and under-staffing of the maintenance team. (Bhatt, K, et al. 2006).

Critical success factors have partly changed in twenty years. Project management and maturity level are more important factors for outsourcing decisions recently, in 2011–2020, than in 2000–2010 (Rahman et al. 2020). Good quality software maintenance requires competence from people involved, and accurate estimation of effort, time and cost. It results in high-quality software, which is dependable, understandable, and efficient and satisfying to

the customer. (Sharon Christa et al. 2017).

There are several maturity models which aim at guiding a company to provide effective and efficient maintenance service. We will have a look at them in the next chapter.

2.6 Maturity models as guidance

Software maintenance can be seen as a service, because the value of it is in the activities which result in benefits for the customer. These benefits are, for instance, corrected faults and new features. This is opposed to software development, which results in a product, namely the developed system, that provides the benefits. The end result for the customer makes software maintenance and development essentially different. (Niessink and Vliet 2000).

Niessink and Vliet (2000) cites Parasuraman, Zeithaml, and Berry (1985) about five identified gaps in software maintenance services, and compliments the list with their own findings on activities to bridge the gaps:

1. Customer's expected service vs. Service provider perceptions of expected service
    - To bridge the gap: Managing commitments with service level agreements and reviewing their accuracy to customer needs with the customer on a regular basis.
2. Service provider perceptions of expected service vs. Service designs and standards
    - To bridge the gap: Planning maintenance services, e.g. staffing, scheduling and identifying possible risks.
3. Service designs and standards vs. Service Delivery
    - To bridge the gap: Tracking maintenance activity to provide this information to a customer, monitor activities and take corrective actions if necessary.
4. Service Delivery vs. External communications to customers
    - To bridge the gap: Handle events in an organised way to ensure appropriate response to all events that occur during software maintenance.
5. Customer's Expected Service vs. Customer's Perceived Service

- To bridge the gap: bridge all the above gaps.

To help bridge these gaps in maintenance service, a framework called IT service capability maturity model (CMM) is established by Niessink and Vliet (2000). Their model targets processes that are key to producing high quality IT services. This particular model has five maturity levels, of which the first is "initial" ad-hoc services, the second is "repeatable" where basic processes are established and past successes can be repeated, the third level is "defined" where processes are documented and standardised and service is tailored for customers, the fourth is "managed" where detailed measures of the process and it's quality are collected and used to control the services, and finally the fifth is "optimising" where continuous improvement is enabled by quantitative feedback and piloting new ideas and technologies.

For example, an organisation on level two implements two processes, according to Niessink and Vliet (2000):

1. Service management: Planning, specification, tracking and evaluation of services
2. Service support: Processes that support the activities to deliver the services, e.g. monitoring of the products and event management.

The benefits of the IT service CMM is its guidance to improving IT service processes. It is quite heavy to implement in its entirety, and it's not meant to be used as a checklist for definite quality of IT services. (Niessink and Vliet 2000).

The IT service CMM is used as one input among many maturity models for a model established by April et al. (2005): the Software Maintenance Maturity Model (SMmm). This particular model aims at providing a context for software maintainers specifically. Their study includes a collection of studies for unique software maintenance activities and a classification of software process areas.

The model consists of four process areas:

1. Process Management
2. Request Management
3. Evolution Engineering

4. Support for Evolution Engineering

(April et al. 2005).

Each of the areas in the model by April et al. (2005) has four to five Key Process Areas (KPA). An example of Request Management KPAs is presented in their study, with detailed description for each goal in the process area. The model has five maturity levels, of which the first is an ad-hoc maintenance process, the third is a state-of-the art maintenance process, and the fifth is admitted to be currently technologically challenging to attain.

The Software Maintenance Maturity Model has proven to be difficult to validate. However, it has been done with a small amount of case studies between the years 1995–2005. (April et al. 2005).

Maturity models propose processes to guide a maintenance service. In addition, a way is needed to demonstrate maintenance effort for customers to provide evidence for value generated for costs invoiced (April 2010). This is usually done with timesheets (Sindhgatta et al. 2010) which we will investigate in the next chapter.

3 Time tracking

3.1 The purpose of time tracking

The relationship between time and money has deep roots in the Western culture - it has gained a status of a social construct. The transformation of time into a commodity is displayed in the world of consultants. Their work is difficult to value, so consulting services are often measured in hourly units called billables. (Yakura 2001).

Likewise to consulting services, maintenance service is often provided on a base of invoicing the time spent by developers doing maintenance activities (Sindhgatta et al. 2010). This requires recording the effort in timesheets. Management support and a time tracking tool is needed so that developers can fill in timesheets, often manually. A detailed report needs to be available for both vendor and customer, who can validate the integrity and credibility of it, and a customer can pay an appropriate payment - although checking validity of hours is tedious for managers, because they usually don't have the hands-on context, and would need to dig into details impractically. (Sindhgatta et al. 2010; April 2010).

Timesheets provide transparency for a customer regarding the provided service, and thus it prevents negative perception for value versus costs. The benefit of reporting hours in detail is in the ease to see trends in time spent in each activity when historical data builds up. The customer can see where the paid hours are spent, and the service provider can see which customer requires the most effort, and where the team could improve in regards to high volume of customer queries. (April 2010).

With detailed timesheet data, a company can identify bottlenecks and difficulties (Shoukourian and Danielyan 2011). Data can reveal team members who are the most and least loaded, or who continuously over- or underestimate their tasks. Discussions with said team members will increase the accuracy of planning and help identify the future scope of work precisely. Time reports can also be used as a control of workload by guiding staffing, for instance, moving resources between projects, hiring and firing, or selling more business (Yakura 2001). Transparent data builds the basis for correct processes for activities that are

part of a time tracking system (Shoukourian and Danielyan 2011).

Literature review reveals several possible use cases for timesheet data:

- Estimate required effort for a maintenance task or process (Hira and Boehm 2018; Chen et al. 2017; Chen et al. 2016; Mellegård et al. 2016; Sneed and Prentner 2016; Choudhari and Suman 2014; Fitzgerald, Counsell, and Peters 2013; Leotta et al. 2013; Kumar et al. 2012; Ricca et al. 2012; Al-ahmadi et al. 2011; Asghar et al. 2011; Friedrich and Bergner 2011; Shoukourian and Danielyan 2011; Cavalcanti et al. 2010; Subramanyam, Weisstein, and Krishnan 2010; Nguyen, Boehm, and Danphitsanuphan 2009; Ooi and Soh 2003; Ruhe, Jeffery, and Wieczorek 2003; Abran, Silva, and Primera 2002; Penny 2002; Rao and Sarda 2002; Bianchi et al. 2001; Krishnan et al. 2000)
- Estimate cost for maintenance tasks (Ivan and Despa 2016; Martínez-fernández et al. 2014; Buchmann, Frischbier, and Putz 2011)
- Evaluate developer productivity (Hira and Boehm 2018, 2016; Shoukourian and Danielyan 2011; Sindhgatta et al. 2010; Aversano et al. 2001)
- Provide data for how much time a maintenance task required (Friedrich and Bergner 2011; Desharnais and April 2010)
- Provide data for overall status of a software project (Napier, Mathiassen, and Robey 2011)
- Provide data for salaries (Bosch 2009)
- Provide data for invoicing customers (Sindhgatta et al. 2010)
- Provide data for studying effect of pressure on total time for delivering a software system (Nan and Harter 2009)

To get an understanding of how a time tracking system might be set up for any of the above mentioned purposes, we review two cases: one by Shoukourian and Danielyan (2011) and another by April (2010).

3.2 Case Shoukourian and Danielyan (2011): Setting up a time tracking system

In this section we examine the study of Shoukourian and Danielyan (2011) about establishing a time tracking process for an operations team. Data gathered from this process allowed teams to work more systematically and made their effort visible to management.

Defining the process started with listing three facts: projects a department was working on, activities performed when working on these projects, and the effort spent on these activities within a project. Knowing this allowed the team to identify bottlenecks and difficulties, increase the accuracy of planning, identify scope of work, and implement processes for identified activities.

Shoukourian and Danielyan (2011) noticed that the list of activities used in time tracking must be limited to an optimal balance between detail and clarity. More precise list of activities gave more precise data, but made time tracking more laborious and less convenient for a consultant. Time tracking was a continuous and never-ending practice, and thus keeping it convenient was important - not only because it would be less expensive if it was easy to do.

Team leads aided in listing activities. This also prepared them for adopting time tracking practices, and to guide their team members with questions. Listing daily activities formally surfaced some problems, like ambiguous responsibilities for a team, or informal and thus untraceable communication methods. The list evolved as the time tracking process was taken into use and real-life activities thought absent did not find their counterpart in the list. Some activities needed to be removed from the list as they were not being used in practice at all.

Dashboards were used to visualise data collected in the time tracking process. Dashboards revealed that operations work could be planned in iterations, and made workload visible per team member. When combined with outputs of different processes, like release testing, the efficiency of teams could be measured: if testing took a lot of time, it was expected to have many defects logged. Dashboard also made visible if a team member spent a lot of their time in other activities than what was their primary responsibility. Table 1 shows activities and

the percentage of time spent on each activity across all projects in a time period.

Table 1. Activities and relative occurrence on time spent across projects in a time period, adapted from Shoukourian and Danielyan (2011, p. 3).

Activity	**Percentage**
Release Testing	50.67 %
System Testing	12.59 %
Meetings	7.74 %
Code and Unit Test	1.47 %
Documentation	1.22 %
Training Participation	0.69 %
Inspection	0.31 %
Performance Management	0.28 %
Inspection Issue Resolution	0.15 %
Investigation	0.06 %
Vacations	24.82 %

3.3 Case April (2010): Categorising tasks for a time tracking system

In this section we examine the study of April (2010) about introducing a categorisation for maintenance activities. With this categorisation, a maintenance provider was able to create a report to understand trends, justify suggested changes to supported software, and get guidance in managerial tasks, e.g. budgeting and staffing.

Time tracking process began by categorising activities so that consultants could choose proper items from a list. Typical maintenance activities found in the study are listed in Table 2. The most interesting categorisation for every request was determining if it was preventive, corrective, perfective or adaptive work, as per ISO/IEC14764 categorisation. This led to an ability to spot trends of work effort in each category. These trends could be analysed for how well they matched what customers expected from the provider.

It was noticed to be a managerial task to ensure that team members tracked their efforts

Table 2. Categorisation for typical maintenance activities, adapted from April (2010, p. 354–355)

Activity	**Definition**
Administrative	General Admin., housekeeping, office work, communication, reporting
Break / Nonproductive	Lunch, tea, personal work, non-productive task, sickness
Classify and update work request	Preventive, corrective, perfective, adaptive or customer request for information
Analysis	Search for causes, replicate/verify
Modification	Code and unit test the change
Regression testing	Verification of fixes and changes
Migration & configuration management	Source code moves, migrations
Reviews	Verification of change, surveillance days following changes
Training	Attending training, presentation & conference
Documentation	Updating technical documentation and D/R

accurately to a time tracking system. To help in this task, a decision tree like in Figure 2 was provided. Accuracy was important because software maintenance service levels were based on it. To ensure reporting was accurate, time recording system and request tracking system were integrated and it was ensured that personnel entered effort only once.

In addition to categorising tasks per type, tasks were to be categorised by a customer and a system in service. This allowed tracking customer profitability, and created understanding in what kind of activities were required for a certain type of software. This aided in the management of software maintenance portfolio.

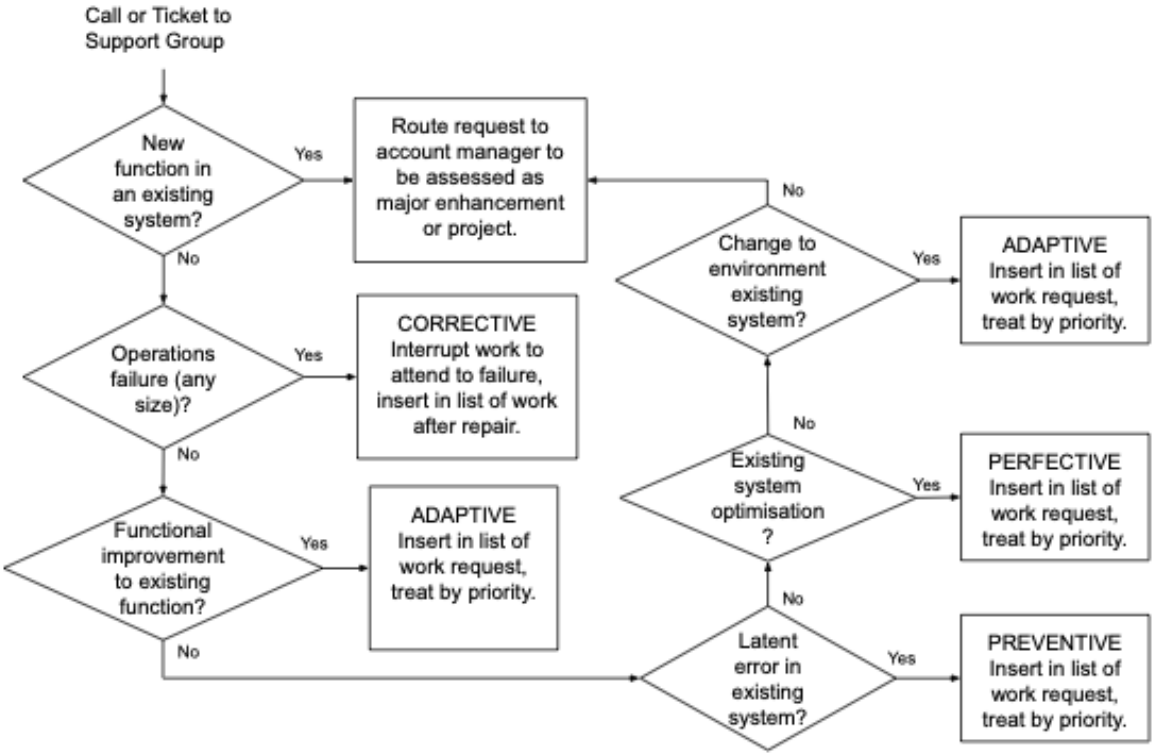

Figure 2. Instructions to assign a change request, adapted from April (2010, p. 354).

3.4 Obstacles and pitfalls of time tracking

In this section we investigate expectations for timesheets and reasons which make time tracking difficult, as described similarly by Yakura (2001) and Sindhgatta et al. (2010), unless otherwise mentioned.

Timesheets are by no means the most exact way of putting a price tag for hours spent. Two distinct hours for one consultant are rarely equivalent, and an hour for two different consultants are certainly not equivalent. There are many tools to facilitate time reporting, but none of them solve the underlying issues of the act itself. It is tedious and error-prone to manually enter time in timesheets. One of the reasons is that developers in maintenance often work on multiple tasks in parallel, for example code reviews and learning activities along with development tasks, or even multiple development tasks in parallel. It is hard to deduce time spent for each activity afterwards.

Accurate data in time reports would be necessary to avoid damaging reputation or revenue,

and misguiding future work. Over-reporting hours will harm reputation if customers notice it. This can happen by lack of specificity in timesheets, if it allows developers to report their actual hours even though their time was spent on unrelated activities, for instance, sorting out unclear expectations or waiting for a third party.

Under-reporting hours causes a company to lose revenue directly with fewer billable hours, and creates a wrong baseline for estimating future work. With inaccurate data, new fixed priced projects might be poorly estimated, and customers might build unrealistic expectations for delivery. If a team tries to meet these expectations, the quality of delivery might be damaged. In addition, April (2010) and Shoukourian and Danielyan (2011) base their trend analysis on assumption of accurate data in timesheets.

In the worst case, timesheets can be used only to make numbers look good. Either a developer can inflate hours to report what is expected, or not report exceeding hours at all - techniques called padding and discounting. Reporting a much different number from the original estimate might cause uncomfortable conversations between a developer and a manager, and developers tend to avoid this. Developers can also be directly ordered to mark hours within some established rules or practices regardless of real time spent. Sometimes developers might feel bad for reporting an exceeding amount of hours because of their own expectations of the time they should have needed to solve an issue or provide a solution.

In addition to invoicing, timesheets can be used internally to track effort and developer productivity. When used as a performance measure, the more a developer bills, the better he or she is seen. Moreover, time is valued differently: customer work is seen as more valuable work, whereas administrative tasks are seen as the least valuable. When there's lots of work, developers have an opportunity to bill as many hours as they want to. When work is scarce, management chooses which developer gets the work, and that causes a cycle of those in demand being seen as better employees simply by virtue of being in demand.

4 Research method

4.1 Problem and research process

The organisation faced the same problems with time tracking that were identified in research. They had detailed categorisation for maintenance activities and there was a need to communicate it clearly to employees to avoid mistakes (April 2010). Time was spent by management to verify time reports and amend wrongly submitted hours (Sindhgatta et al. 2010). The complexity of time tracking was well known in the case organisation. For example, verifying validity of hours resulted in several recalled reports weekly due to wrongly chosen assignments, and reminding employees to submit hours in the first place.

This study created new instructions in the form of a decision tree, in addition to existing instructions the case organisation already had. Both new and old instructions aided in task categorisation for time reports. This study evaluated the perceived usefulness of new instructions within a software maintenance team. The aim of the research was to help employees categorise tasks correctly.

The research process started with literature review, which inspired a mapping exercise with the team. The data from this exercise was used to create instructions in the form of a decision tree. Instructions were given for the team as additional help for categorising tasks for the time tracking system. Finally, interviews were conducted regarding software maintenance, time tracking, and usefulness of the instructions. The process of research is depicted in Figure 3.

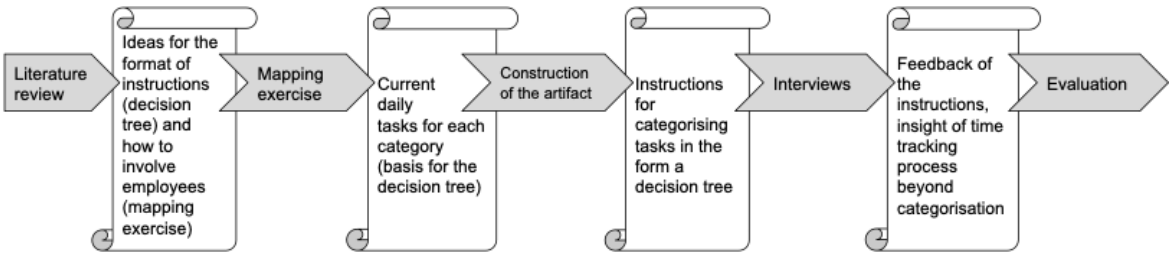

Figure 3. Research process, and the output from each of the step during the process

4.2 Research method

We used the Design Science Research (DSR) method. DSR is an iterative research and design process to solve an important problem (Hevner et al. 2004). Another guiding method for this research was Constructive Research Approach (CRA). It is introduced by Kasanen (1993) with the core of solving a practically relevant problem in collaboration with practitioners.

We did not see conflicts in following both of these methods. CRA can be seen as a subset of DSR with relevant distinction for this study that DSR puts more emphasis on evaluating produced artifacts than CRA does (Piirainen and Gonzalez 2013). Both methods proceed from understanding the problem towards possible solutions and developing and evaluating one of the solutions, as can be seen in Figure 4.

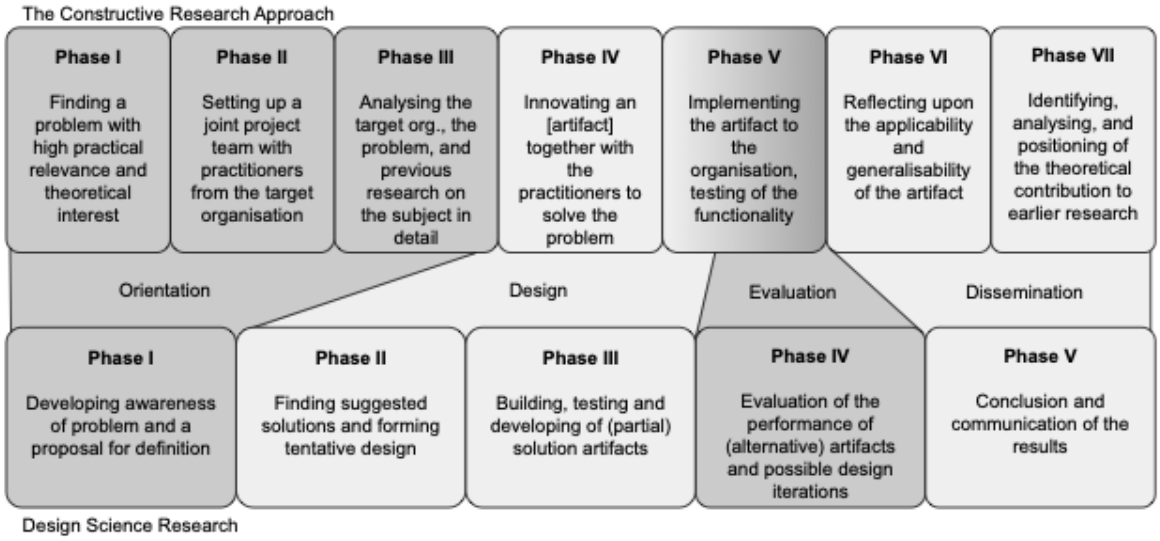

Figure 4. Process comparison for Design Science Research and Constructive Research Approach, adapted from Piirainen and Gonzalez (2013, p. 211)

We chose DSR because its goal is to increase knowledge and understanding of a previously unsolved or sub-optimally solved and important business problem so that development and implementation of new technology-based solutions becomes possible for the problem by creating an artifact (Hevner et al. 2004).

Keeping time tracking convenient is important because it is a never-ending activity and less

expensive if it is easy to do (Shoukourian and Danielyan 2011). The organisation had experienced the time tracking process being inconvenient. Thus, this problem was important and sub-optimally solved in this organisation. It fulfilled the requirement for a problem as is described in the second DSR guideline (Hevner et al. 2004).

Close collaboration with practitioners was an inherent part of how the research was conducted. The author was employed by the organisation for which the research was done. Thus the core of CRA, collaboration with practitioners (Kasanen 1993), was strongly present in this research.

4.3 Literature review

Problem awareness was built on the author's experience and supplemented by studying previous research. The goal was to build a general and comprehensive understanding of the topic (Kasanen 1993). The purpose of this thesis was to aid the time tracking process in software maintenance context. To be able to propose a solution, the terms and practices were first defined for software maintenance and time tracking.

To start finding literature, we first used Google Scholar with search words "software maintenance" and "timesheet", and search phrase "Filing timesheets in software maintenance". This allowed us to find some initial articles to fine-tune search words. In the result, "timesheets" got an alternative of "billables" and "software maintenance" got an additional term "maintenance service". Search criteria for search words were limited to articles published after the year 2017 to include at most five year old papers due to rapid evolving of the IT industry.

When suitable articles were found, we reviewed both references and citations to find new relevant articles - the technique known as snowballing. This led to source material circling around the initial question. Suitability of an article or other source material was defined by its validity to software maintenance field, describing software maintenance or the the tasks included, or describing time reporting in IT consultant context. Articles diving into technical implementations of systems managing maintenance workload or tasks were rejected because they did not answer questions about time tracking practices. Academic articles, conference papers and standards were accepted as source material. To decide whether to include a paper,

we read the abstract and if still in doubt, read the rest of the paper to find out if it was suitable for this research.

The databases used to search for articles, and to retrieve articles found in snowballing, are: ACM Digital Library, Google Scholar, IEEE Xplore, ProQuest, SAGE Journals, SpringerLink, Taylor & Francis Online, and Wiley Online Library.

To make an overall inspection of research around the topic a systematic search was conducted in aforementioned databases. The purpose was to find papers regarding either timesheet usage or time tracking process within software maintenance. Search clauses for two separate searches were:

1. timesheet AND "software maintenance"
2. ("time tracking" OR "time reporting") AND "software maintenance"

Search was conducted between 8th and 12th June, 2022, except for Google Scholar on 14th, 20th and 21st August, 2022. As Google Scholar gives hundreds of results, only the first four pages were examined, and thus 40 results total for both search clauses. Suitability was determined by investigating if an article answered any of the following questions: "How to communicate effort in timesheets?", "How to categorise time in a time tracking system?" and "How to make it easier to report time?". Articles before the year 2000 were rejected due to the rapid evolution in the IT industry. If a resource wasn't freely available through the university login, it was not accepted as material.

Results of the systematic search are described in Table 3. With the second search clause in SpringerLink all efforts to limit the search resulted in an error "Bad Message 431, reason: Request Header Fields Too Large". Thus the size of the result set is so large and yet there are no selected articles.

As the scarcity of selected articles show, time tracking and reporting in software maintenance context is not widely researched. Most of the databases did not find many articles with given keywords. Google Scholar was an exception but it remains unknown if further results after four pages would give the same portion of selected articles. Overall, roughly third of the results were selected. From those, only a handful contributed to time tracking from the

Table 3. Search result amounts for systematic database search. *Limiting results for SpringerLink gave an error, and thus search could not be conducted.

	timesheet AND ...		("time tracking" OR...	
Database	Results	Selected articles	Results	Selected articles
ACM	10	4	24	5
Google Scholar	377 (40)	17	518 (40)	12
IEEE Xplore	0	0	5	1
ProQuest	3	1	18	4
SAGE Journals	3	0	2	0
SpringerLink	1	0	3688	0*
Taylor & Francis Online	3	0	7	0
Wiley Online Library	9	1	21	2

process point of view. Majority were researches using timesheets as source data for different measures.

Quality of found research papers is good because they are obtained from respected databases. Quality of search clauses cannot be proved by this research only. Maybe there were other articles which used slightly different keywords. However, these searches found two key articles the author had found previously, namely April (2010) and Sindhgatta et al. (2010), and one new key article, Shoukourian and Danielyan (2011).

Reliability of the search results relies on the understanding and any bias the author might have towards the topic and how she interpreted articles at the given moment. Results of the systematic search are listed in Chapter 3.1 but the reader cannot derive from which database each result is coming from. However, search is repeatable because the search clauses and databases are listed.

4.4 Mapping exercise

Both April (2010) and Shoukourian and Danielyan (2011) identified the need to standardize software maintenance activities in an organisation. The latter also provided a comprehensive

description for how this standardisation was done, as described in Chapter 3.2. Following this process the author asked her whole team to map their daily tasks to current task categorisation given by the employer. Thus knowledge of theory enabled this step in the design process (Peffers et al. 2008).

The exercise was held in a remote team meeting and facilitated by the author in an online Miro board. During thirty minutes, the attendees filled in a total of 67 sticky notes individually in a template pre-populated with the current categories and one additional column for an "Unknown" category. Sticky notes written by team members included actions like "Customer meetings", "Responding to invoice queries" and "Pairing up/rubber duck debugging".

Ten team members attended the exercise. It was voluntary and attendees could choose to answer anonymously. Data gathered this way was somewhat unique, because it would produce slightly different results if the exercise would be repeated again. Attendees relied mostly on their recollection of their daily activities. However, daily activities were a fact and thus answers wouldn't differ much in the second workshop.

4.5 Construction of the artifact

Reusing the example from April (2010), the author created instructions in the form of a simple decision tree by combining meaningfully all the listed activities from the mapping exercise. To make the list complete of activities in addition to those listed by the team, she gathered tasks listed in the existing time reporting instructions. The decision tree became sixty items long, with every decision symbol suggesting an activity and directing the user to either a correct category or to the next decision symbol.

The tree was the artifact produced in this research. It is described in more detail in Chapter 5.

4.6 Interviews

To understand how the artifact was perceived, and to understand the problem domain better, semi-structured interviews were conducted. It took four weeks to interview chosen team

members. The first person was interviewed after the artifact had been available for two months. The delay was partly due to summer holidays, and partly to allocate time for team members to use the decision tree. The lengthy period for using the instructions was allowed because time tracking was mandatory in a one week cadence, so two months enabled team members to use the instructions about nine times at minimum.

The author interviewed eight team members individually. She interviewed three developers and three senior developers, one junior application specialist and one application specialist. Among these were three squad leaders responsible for three to six person teams and their respective customers. Five of the interviewees had attended to the mapping exercise.

The shortest tenure within interviewees was under six months and the longest almost four years. Average tenure at the time of interview was 1,5 years and median one year and four months. The shortest interview lasted for 23 minutes, and the longest 52 minutes. Average interview length was 32 minutes and median 26 minutes. Interviews were conducted and recorded as remote meetings in Google Meet in the Google Workspace of the author's employer and later transcribed by the author. Transcriptions were used to analyse the answers.

Interviews were conducted mainly in English, although it was not the native language for any of the participants nor the author. When Finnish was the native language for the interviewee, Finnish was used. Participants lived in Finland and other European countries, and came from six different cultures. This could have affected how interviewees communicated and expected others to communicate. The author considered these concerns minor due to having worked together in the same team and using English as everyday business language for a long time with the interviewees. Finnish was the author's native language.

The author had short questions prepared for the interviews, and then trusted the conversation to continue seamlessly. She also asked for clarification when she did not fully understand, or asked an interviewee to tell her more. This gave room for interviewees to explain themselves in their own words. Even though discussions were unique, a prepared template with questions would guide a repeated interview into the roughly same direction and thus answers are reliable. Prepared questions can be seen in Appendix A.

The author was a supervisor for five of the interviewees and a leader for all of the interviewed

team members. This posed a question of power balance and how honest in their answers interviewees felt comfortable to be. The author had a reason to believe the interviewees were honest in their responses and data gathered was well qualified. Time tracking was a frequent topic in team communication in the form of instructing how to do it and criticising how complex it was to do. From the author's side she felt the relationship with interviewees was egalitarian, decisions were made seeking consensus, and she could confront each other professionally with all of the interviewees. The author had at least weekly collaboration with most of the interviewees and at least monthly with all of them. This ensured open collaboration in interviews, which is one key aspect to quality data gathering in the evaluation phase (Piirainen and Gonzalez 2013).

4.7 Evaluation

Evaluation of the artifact, the decision tree, was done once with the help of answers from interviews after all interviews were conducted. Interviewing is classified as an observational evaluation method by Hevner et al. (2004), and it suggests that the artifact is examined in depth in the business environment. Among the evaluation methods Hevner et al. (2004) propose, it is the best suited for this research. To compare, experimental or descriptive methods weren't needed because there was a real environment to test the artifact instead of an artificial environment or written scenarios, and we did not need to create arguments based solely on past research. Testing and some forms of analytical methods weren't suitable because the artifact was not an information system but instructions for a process.

A question can be raised for how well an interview suited to evaluate claims from literature regarding time tracking in software maintenance, and how it evaluated the utility of the artifact. This study was a qualitative research and the interviews verified many observations from literature. The perceived usefulness of the artifact was made clear by the interviewees, albeit the small size of the case team.

Albeit the research question seeking several kinds of instructions, only one solution candidate was produced. The solution candidate wasn't improved iteratively: it was built and evaluated once. As DSR is inherently iterative, being a search process for a design (Hevner

et al. 2004), this guideline wasn't followed in this research. To respect a meaningful scope for this study only one iteration for one solution candidate was done. The iteration already included three actions: the mapping exercise, construction of the instructions, and interviewing users. The results of these actions provide the first step for further studies.

With current knowledge, some of the interviews should have been done before the mapping exercise and construction of the instructions. It was also the suggestion of Kasanen (1993) to gain general and comprehensive understanding on the topic. The viewpoint and experience from the author and ideas from literature alone did not count as comprehensive understanding. Interviewing team members early would have provided more information on the problem domain and could have changed the plans for solution candidates completely.

The intrinsic expectation that evaluating the artifact's utility would prove the attached theory to be valid can be questioned (Piirainen and Gonzalez 2013). The linked theory in the artifact is the need for managerial attention and clear instructions in the form of a decision tree (April 2010), and inviting team members to categorise activities to prepare them for time tracking practices (Shoukourian and Danielyan 2011). These theories can be proved in one use case within this research: when a new employee learns time tracking practices.

5 The Artifact: Instructions for task categorisation

The artifact designed and constructed in this research is instructions in the form of a decision tree guiding in categorising tasks correctly in time reports. The form of the decision tree was chosen to imitate instructions described in April (2010). For the author, it felt comfortably logical for a developer like herself. It was a styling chose, and style is an inherent part of design (Hevner et al. 2004).

The tree had sixty decision symbols, each pointing to a correct category or to the next decision symbol. Categories were highlighted with different colors, so the user could recognise a category from the color without reading the label. Parts of the tree can be seen in Figures 5 and 6.

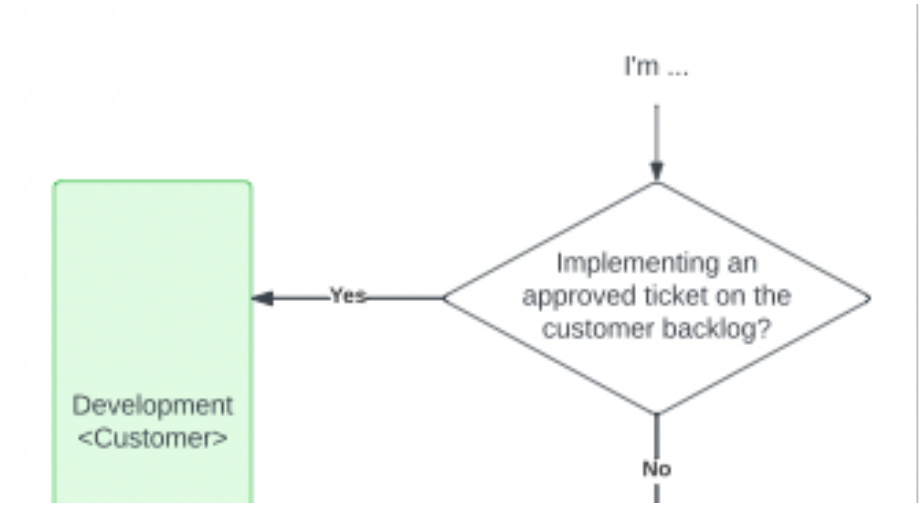

Figure 5. The beginning of the decision tree stating how a development task should be categorised.

The tree was structured logically by combining tasks related to similar activities and labelling the group on the right of the decision tree. These groups were, for example, Development, Customer meetings, On-boarding, Estimation, Incident management, and Learning.

Some answers in the tree had further conditions because a task might belong to several categories depending on its initiating force. In such a case, a question of a trigger for the task was asked, and each trigger pointed to the correct category. For example, an internal

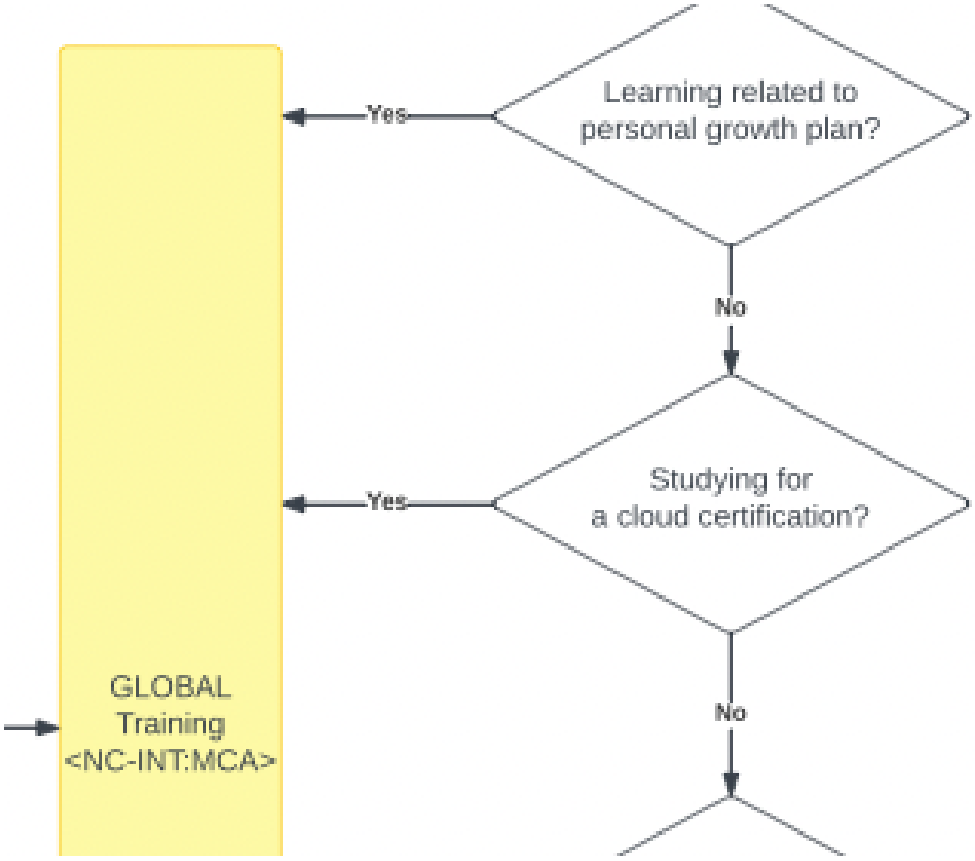

Figure 6. A part of the decision tree, stating how personal development and training activities should be categorised.

meeting might have been held because a new team member was introduced to a project, or obstacles needed to be solved to continue developing an approved ticket. Time spent would be categorised differently in these cases.

Triggers can be seen in Figure 7. Lines leaving from a solution are visible, with a trigger specific to each line. These lines pointed to solutions in the tree at the same level where a similar activity was defined. Such an arriving arrow from the left can be seen in Figure 6. This structure was aimed to build meaningful connections between related activities and help in task categorisation in the future, as a trigger was questioned and the user was guided to a similar activity.

The reason for including so many tasks in the decision tree was to decrease abstraction level and aid in categorising a task by the help of examples. Existing time reporting instructions

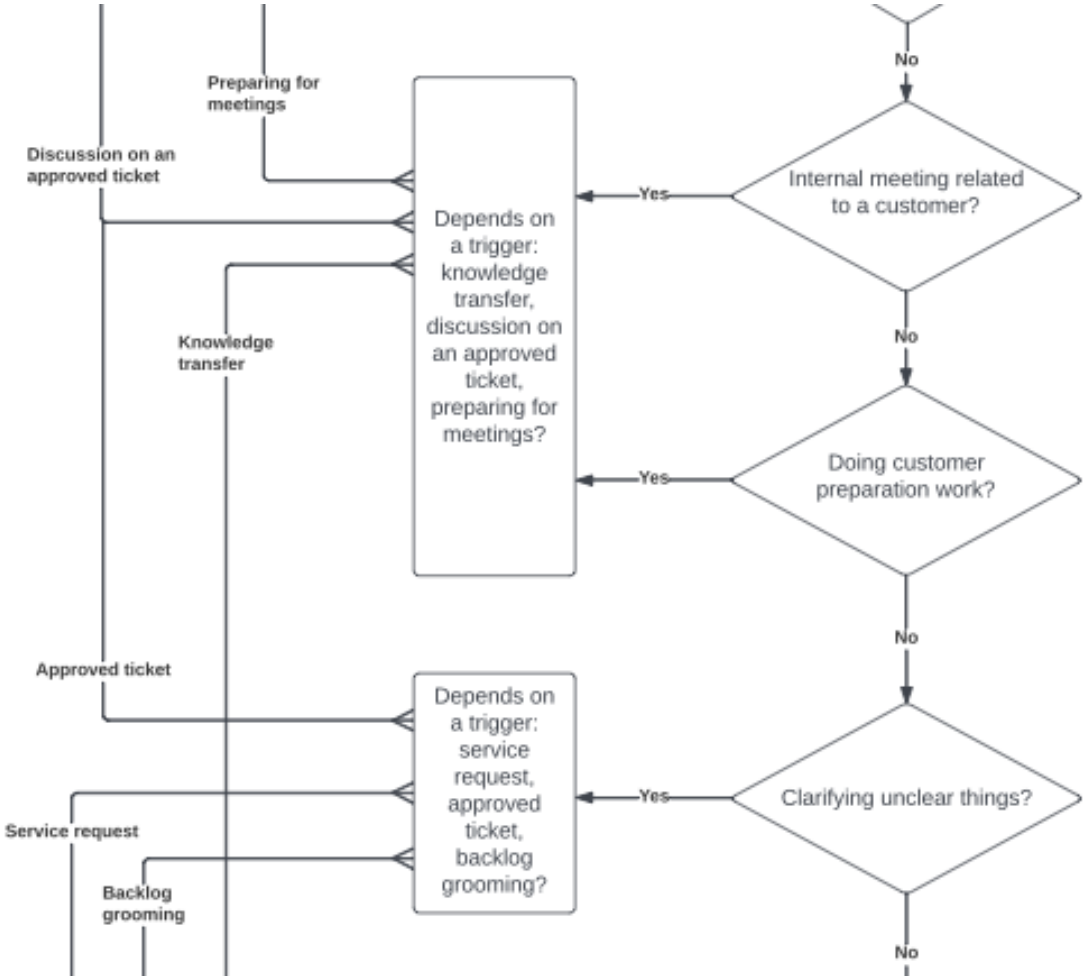

Figure 7. A part of the decision tree showing how triggers are used to further categorise a task

of the company had higher abstraction level and only some tangible examples of activities.

6 Results

6.1 Output of the mapping exercise

Before the artifact was created, a mapping exercise was conducted with the team. In the exercise, the team members were asked to map their daily activities to current time tracking categorisation. The exercise resulted in 67 individual sticky notes with an activity on each note. The amount of tickets revealed the multitude of activities a developer might have in a maintenance team. Employees with short tenure experienced the exercise useful as it answered some of their questions regarding task categorisation when they saw how other team members mapped the activities. The following quote, as all quotes in this chapter, are from interview transcriptions. Fill words are removed when they did not bring more value and only hindered readability.

> It [the mapping exercise] was very useful from my perspective. It helped me to better define the hours, or it answered some of my questions where to put something. (Application specialist #1)

The activities written by the team were used by the author to build the artifact. During the mapping exercise, some activities were mapped wrong by team members. The author also included these activities in the tree and mapped them to correct categories. She did not highlight wrong mapping, but enforced correct mapping by including the activity in the tree. She disregarded one activity because it wasn't relevant for the team and she did not want to highlight its importance by including it in the decision tree. For the same reason, and to protect individual privacy, we don't tell more about the discarded activity in this study. As irrelevant activities go, "going to gym" was written by the team and this was a reminder for the author to include a task of longer breaks and respective categorisation in the tree.

The activities in the tree followed closely the definition of Chapin et al. (2001): there were activities regarding modifications, e.g. changes in source code or documentation, and other activities, e.g. evaluating software and consulting. Only customer training was not among the activities in the decision tree. Besides these types of activities which Chapin et al. (2001)

listed, it was also visible the team was working on many activities seen as part of successful maintenance service by Bhatt, K, et al. (2006) and Sharon Christa et al. (2017), e.g. training employees or communicating requirements with a customer. Examples can be seen in Table 4.

Table 4. Activities identified in the literature compared with the results of the current study

Activity in the literature	**Activity listed in the current study**
Activities by Chapin et al. (2001):	
Change in source code	"Actual development tasks with tickets approved"
Change in documentation	"Updating a runbook regarding an implemented ticket"
Consulting	"Discussing with a colleague about a task or issue in the project and try to find a solution"
Evaluating software	"Onboarding a project"
Activities by Bhatt, K, et al. (2006):	
Training employees	"Studying for a cloud certification"
Knowledge transfer with a customer	"Customer communication about incidents"
Activities by Sharon Christa et al. (2017):	
Communication of requirements with a customer	"Daily, weekly, bi-weekly or monthly meeting with customer, or preparation for it or writing a memo afterwards"
Knowledge transfer within the team	"Transferring or receiving knowledge inside the team in case of new team members in a customer project"

6.2 Usage of the instructions

This study set out to find what kind of instructions would aid in the time tracking process. The produced artifact was one kind of instruction which was evaluated with the help of

interviews.

The result of this study is that the decision tree aided when a new employee learned the rules of task categorisation. The tree was perceived as easy to use and answered many questions a newcomer might have.

> I have used it [the decision tree] practically every week. (Developer #1)

> There is quite [a] lot of information [in the tree] and that's a really good thing because we try to cover as many scenarios as possible. Everything is included and anyone who's trying to figure it out, it's easier for them. And it's actually well structured, so also[a] good thing. I think after a year [..] you can memorise this. (Developer #4)

The tree was very detailed and that was perceived as beneficial. The tree covered many activities and edge cases, only missing some administrative tasks. It was experienced to be logically structured. Labelled groups on the right were found useful for skimming through the tree. Occasionally defined triggers for certain activities were thought to aid in clarifying principles of time tracking and building a lasting mental model of the task categorisation rules. For an employee with short tenure the tree answered many questions about categorisation. An employee with longer tenure still found the tree useful to shorten the time to find answers to edge cases.

> So first I usually check the right part, for example on-boarding, or knowledge transfer, and then I look [at] those diamonds. If something I want to describe is there [..] then I see on the left [for] these squares. So that's the way I'm proceeding. (Application specialist #1)

The tree revealed the complexity of time tracking in a tangible and even surprising way. Employees admitted they were not thinking about the many peculiarities of time tracking after they had built a mental model of the rules. Neglecting details in everyday task categorisation did not prevent them from submitting correct time reports. A possibility to misuse the tree by following the rules to the letter and avoiding any creativity in categorisation was concerning for some employees. It was suggested the information in the tree would be used for an

algorithm which would categorise tasks automatically given an input. A program using the algorithm would reduce the need to transfer the knowledge in the tree to every employee separately.

> I feel that when people try to follow processes step by step without giving it much of a thought, it limits how somebody can react. I also feel that it doesn't improve one's ways of working if he's just [following the protocol blindly] without giving it much of a thought. (Squad leader #2)

6.3 Insight of the time tracking process

Time tracking was perceived to be the most annoying part of the job. The purpose of time tracking was recognised but it was seen as the "mandatory evil," non-trivial activity. Time tracking process was compared to a field of art, as employees needed creativity to figure out the correct category for an activity, and on top of that favouring customer work related categories whenever possible. Employees did not see ways to influence time tracking for the better, so they were just trying to live with it.

> Filling in timesheets is - well, it is not that difficult for now, I'm used to it, but it still feels like [an] extra thing to do, so it's a bit annoying. (Developer #4)

> When the others advised me, they made it [time tracking] sound easy, but for me it was extremely confusing - I don't remember why. (Squad leader #3)

Time tracking was seen to get easier by time, as the mental model of categorisation rules formed. However, even after one and half year tenure a developer was still making occasional mistakes with categorisation. The decision tree aided in learning the categorisation and building a mental model of the categorisation rules. After employees had created a mental model for categorisation, using the tree was not seen as sustainable anymore. It would have been too slow to find every task from the tree every time it occurred. Instead, the focus shifted to other problems in time tracking, namely, reporting non-billable hours, exceeding time estimations, and recalling tasks done throughout a day or a week. We will examine these three problems in detail in the following paragraphs.

Reporting billable hours was considered as an essential part of consulting business, and it was even recognised as a key performance indicator. Developing internal tools was perceived to be respected, but organisational and administrative tasks, or tasks related to thinking, learning and conversations, were experienced to be hard to categorise. A developer had a feeling of being frowned upon if they reported too much time on categories related to internal work.

> I feel there is a push towards billable assignments, the customer work. I do feel that those are valued more highly. (Squad leader #2)

Exceeding time estimations occasionally caused developers to feel guilty. They felt pressure to stay within an estimated time frame and to report only the estimated amount of hours to a customer work related category. If they had exceeded the estimated time frame, they sometimes hid the exceeding effort by distributing it to other, mostly internal work related categories. Estimating a needed time frame in advance was seen as difficult due to the nature of software maintenance work, as the systems in service were not thoroughly familiar for the developers. Learning to do better estimations was expected to require a feedback loop of following realised time spent and comparing it to estimations, and this was lacking for many systems. Regardless of the balance for estimated and realised effort, the employees experienced the company was encouraging them to report the actual working hours honestly.

> For sure, sometimes if this is supposed to take [a] few hours and you spend half a day, you kind of feel bad but still that's the amount of work it took to complete, so I'm putting down that. (Developer #2)

Recalling tasks done throughout a day or a week after time had passed was seen as really difficult or even impossible. Developers were serving several customers and working often with several tasks in parallel. Developers acknowledged the need to keep prompt notes on tasks at hand and the time that was spent on a task during the day. However, note keeping was seen as an additional task which needed to be remembered to do, and some employees struggled with it. Tooling was seen related to the problem of following promptly with a task and the amount of time it took. The current time tracking system was experienced to be cumbersome to use and missing a timer feature for easily tracking time spent on an activity.

Developers thought that if they were able to choose the tool of their choice they could be more timely and accurate with their time reports.

> If you don't do it [time tracking] straight away it's often a problem of simply recalling what you did. (Developer #2)

> It can be that I did fifteen minutes this, half an hour that - ten different assignments per day. Sometimes we have ideal weeks so that I can focus mainly on one customer's development tickets. (Developer #1)

> There are a lot of people who use 3rd party tools to manage their own time, and then they move this to [the time tracking tool]. I would say that maybe figuring out how to integrate 3rd party tools with what we have [...] would allow people to track their own time more accurately. (Squad leader #2)

7 Discussion

7.1 Summary

The research problem was to study what kind of instructions would aid in the time tracking process for software maintenance service. It was found that detailed instructions for categorising activities was helpful for new employees. The level of detail was concerning only in the case it would guide employees to follow instructions to the letter and avoid any creativity. Additionally, a major revelation rose unexpectedly from the research: time tracking was perceived as the most annoying part of the job.

7.2 Mapping exercise

To begin with a time tracking system, listing activities in software maintenance service was recommended by Shoukourian and Danielyan (2011). It was said to prepare team members for time tracking practices and to reveal problems, like untraceable communication methods. In the case study, the exercise did not raise awareness of any problems within current activities. The exercise was useful for creating the artifact because the team provided so many examples of activities to be included. The exercise itself had value for employees with short tenure. However, as the tree had all the same activities that were produced in the exercise, it would be enough for new joiners to use the tree only and not participate in another mapping exercise.

7.3 Instructions

The artifact, the instructions, followed the same format as was presented by April (2010). Another possible way to format and present instructions would have been a table with categories in the first column and possible actions belonging to the category in the second. The table would possibly have worked with the same data that was used in the decision tree, although the question would have been presented in the opposite order - instead of thinking about the task at hand and finding a category for it, the user would have needed to think: "If I

use this category, my task should be similar to those listed here". The table could have been implemented in the same web page as the previous instructions in the organisation were.

The second alternative for the decision tree would have been a format of "Frequently Asked Questions (FAQ)". The activities could have been the questions and categories the answers. The FAQ format might have been better than the table because it could have been arranged to similar sections as the tree was structured, for example headlines regarding on-boarding activities or customer backlog related tasks. Under the headlines the activities could have been formatted to descriptions of a situation where such an activity would have happened. This format would have been easy to embed in the existing web page for instructions.

Presenting the information in a web page instead of presenting the tree in a PDF file would have supported faster search functions from a web browser compared to a PDF file. Also, the instructions would have been all in the same page as compared to now two different sources. A benefit for the PDF file was that users could download it to their own computer for faster access.

With additional program supporting interactions, the decision tree format could have been taken further. The first question could have asked for the group of activities and then revealed the activities within that group only. This approach would have hidden the size of the tree and possibly made it more appealing to use.

In practice it was experienced that the tree lacked examples for administrative tasks and communication. Employees faced difficulties in categorising time spent in reading and replying email or instant messages, and tasks regarding thinking, learning and preparations. The decision tree did not guide in building a mental model for categorising these activities to neither customer nor internal work. An iteration for the tree would have been necessary to document the context the organisation wanted employees to tie these activities into.

The tree was made very detailed to be the opposite of the rather abstract instructions the company previously had. While doing this, an advice for finding a balance between detail and clarity from Shoukourian and Danielyan (2011) was forgotten. As they stated, a precise list of activities would make time tracking more laborious, and that would not be ideal since time tracking was a never-ending activity. However, the decision tree did not increase the

amount of categories for time tracking - it only provided more examples for categorising activities.

The detailed tree provided enough examples for new joiners to find matching activities to tasks they were reporting. For this use case, the artifact was useful and helpful. We can say it had utility (Hevner et al. 2004). The artifact would continue to provide correct answers as long as the underlying categorisation wouldn't change. In case the categorisation rules changed, the tree should have been updated accordingly. As the tree was not implemented in the same web page as the existing instructions in the organisation, it would be less likely to be updated. Thus, the sustainability, in other word the fitness (Gill and Hevner 2011), of the artifact was not certain. To ensure needed updates, the information in the tree would have been better to implement in the web page.

7.4 Time tracking process

The artifact lost its utility when an employee had learned the categorisation rules. The problem of time tracking changed when the rules were memorised and thus the same instructions did not aid developers anymore. Developers felt stress due to unclear context for administrative tasks, and when they needed more time to finish a development ticket than they initially estimated. The organisation was perceived to value customer work over internal work, but to avoid disappointing customers with an expensive bill, developers under-reported their hours to customer related work and over-reported them in internal work categories when in doubt of context or exceeding the estimated time frame. Under-reporting was called padding by Yakura (2001), and she, too, witnessed developers being ashamed of billing the whole time in case they thought they should have implemented a solution sooner.

Another source for stress was the need to keep a note of ongoing tasks for time reporting. Developers were working with multiple tasks parallel or interleaved, and found it hard to separate time spent for each activity afterwards. This is exactly as Sindhgatta et al. (2010) observed software maintenance work to be: developers working with many tasks parallel and facing difficulties in separating a day in thirty minute increments. Keeping prompt notes is not solvable by the same instructions that aid in categorisation - the problem in this case

would be of not knowing the activity in the first place. For this problem, entirely different instructions or even managerial intervention might be needed. As April (2010) notes, it is a managerial problem to ensure employees track their time accurately.

7.5 Limitations

This study has several limitations. First, the focus is on one team within one organisation. The contribution might be generalisable to other software maintenance teams based on the similarities within the results from the interviews of the target team and research in the matter. However, generalisability was not the goal of this research, but rather providing in-depth knowledge of time tracking practices in a software maintenance team.

Second, only one solution candidate and one iteration was done in this study. Thus, this study cannot comment on how other kinds of instructions would aid in time tracking besides the constructed artifact. The results from this research are the first step in a larger process to understand the problem domain and find solutions for it.

Third, the information was gathered via interviews. There was a notable power imbalance in the interviews due to the role of the author as a supervisor or leader for the interviewees. Despite the power imbalance, interview answers demonstrated uniformly how the artifact was used and how time tracking was perceived.

This research validated some observations from previous research and contributes one case study to the so far scarce research of time reporting in software maintenance context.

7.6 Further research

To answer the question of what kind of instructions would aid in time tracking, more solution candidates and iterations would be needed. At least two separate problems were revealed in this study: to learn the time tracking rules, and to infer the context for activities. The further research could seek suitable instructions for these problems.

Further research could address the managerial problem of guiding employees so that they

did not need to feel guilt for time they spent on activities. Effect of a feedback loop from estimated and realised time frame, or feedback from customers regarding time reports could be investigated.

Further research could study possible solutions for keeping track of tasks throughout a day. Possibilities to aid in note keeping via coaching or tooling could be a path for further research. It would be interesting to study how longer than half an hour increments would affect customer invoicing and employee experience of time tracking.

7.7 Implications for practice

1. Time tracking is tedious, as was confirmed in this research in the context of a small case study. It's a never-ending activity (Shoukourian and Danielyan 2011) and it takes time to learn it. This study found that detailed instructions with many examples of real life activities help in learning the time tracking rules. Give yourself the time to learn the rules.
2. The need to seek guidance from instructions was seen to lessen within time in this case study. The time tracking rules were learnt and developers were able to deduce a category for an activity by heart. Aim to understand the principles of time tracking and task categorisation to enable categorising tasks easily without instructions.
3. It's difficult to separate time spent for activities afterwards (Sindhgatta et al. 2010). In this case study it was suggested to find your own way of keeping notes on a task and its duration as an inherent part of the task. Seek support from peers and management to find alternative tools and practices.

7.8 Implications for management

1. A company could avoid losing money by supporting developers in reporting their hours correctly and avoiding under-reporting. This study confirmed the finding of Yakura (2001) about developers under-reporting their hours when they felt guilty or unsure of justification for reporting hours in customer work categories.
2. Detailed instructions were seen to aid in learning time tracking rules in this research.

However, the instructions weren't always enough to aid in learning the principles for deciding a category between internal and customer work for those activities that were not directly related to customer requirements, for example thinking and learning. Managers might need to coach developers in deducing the right context for an activity despite having detailed instructions.

3. If reported hours are not accurate, it could result in incorrect future estimations and too high customer expectations (Sindhgatta et al. 2010). Coaching developers in correct categorisation might reduce under-reporting and ensure correct baseline for future estimations.

8 Conclusion

This study set out to discover what kind of instructions would help time tracking in software maintenance context. Software maintenance is a neglected part of the development life cycle in research (Sharon Christa et al. 2017). The literature review in this study reveals how time reports are used as source data in several research papers. In practice, time reports are used for invoicing customers and estimating future work (Yakura 2001; Sindhgatta et al. 2010). However, time reports were seen subject to inaccurate data (Yakura 2001; Sindhgatta et al. 2010)and there was a gap in literature exploring how companies can ensure and increase the accuracy.

Based on a design science research in a consultancy company we found that detailed instructions aided employees to initially learn the categorisation rules of activities for time reporting. The detailed instructions were presented as the artifact of the study. The artifact was instructions in the format of a decision tree, aiding a user to categorise an activity with sixty examples of activities and their correct category. The data for the tree was gathered from the team in an exercise where team members could write any of their daily activities and map them to current categorisation.

The tree had utility because it was perceived as an useful aid when learning to categorise tasks initially. Its use was not seen sustainable after about a year because reporting time was faster when a developer remembered task categorisation by heart. Different solution candidates would be needed to solve problems in time tracking after learning the categorisation.

The research revealed unanticipated problems regarding time tracking. After memorising the categorisation rules, the hardest thing was perceived to relate to justification of used hours on an activity: developers felt guilt when exceeding the estimated time frame for a customer work item, and when they used considerable time on internal work instead of customer work. Another problem was related to simply recalling daily activities. In software maintenance, developers usually worked with several parallel and interleaved tasks, so it was hard to separate time spent on each task afterwards (Sindhgatta et al. 2010). Necessity but tediousness of keeping prompt notes arose in this research. All in all, time tracking was

perceived as the most annoying part of the job.

Our research contributes to the scarce research of time tracking processes in software maintenance by documenting experience in one company and showing how detailed instructions can help to learn task categorisation rules for time reporting. We confirmed additional obstacles in the time tracking process that were previously identified by Yakura (2001) and Sindhgatta et al. (2010). The results cannot answer these obstacles but build the first step for further research in the topic.

Further research is encouraged to propose more solution candidates for instructions to aid in time tracking process. This research gives basis for developing detailed instructions further and understanding the problem domain better to address, for example, note keeping habits or experienced guilt in time reporting.

Bibliography

Abran, Alain, Ilionar Silva, and Laura Primera. 2002. “Field studies using functional size measurement in building estimation models for software maintenance”. *Journal of Software Maintenance and Evolution: Research and Practice* 14 (1): 31–64. ISSN: 1532-0618. https://doi.org/10.1002/smr.245.

Al-ahmadi, Haneen, Rodziah Atan, Abdul Azim, Abd Ghani, Masrah Azrifah, and Azmi Murad. 2011. *Agent-Based Cmmi for Software Maintenance Process Measurement Model.* Visited on August 21, 2022. http://worldcomp-proceedings.com/proc/p2011/SER2172.pdf.

April, Alain. 2010. “Studying Supply and Demand of Software Maintenance and Evolution Services”. In *2010 Seventh International Conference on the Quality of Information and Communications Technology,* 352–357. https://doi.org/10.1109/QUATIC.2010.65.

April, Alain, Jane Huffman Hayes, Alain Abran, and Reiner Dumke. 2005. “Software Maintenance Maturity Model (SMmm): the software maintenance process model”. *Journal of Software Maintenance and Evolution: Research and Practice* 17 (3): 197–223. ISSN: 1532-0618. https://doi.org/10.1002/smr.311.

Asghar, M Zubair, Imran Ali Khan, Waqas Anwar, and Bashir Ahmad. 2011. “Systemized Approach for Software Corrective Maintenance Effort Reduction”, 8. Visited on August 14, 2022. https://www.academia.edu/download/63213694/J._20Basic._20Appl._20Sci._20Res.__201101356-1362__202011.pdf.

Aversano, L., S. Betti, A. De Lucia, and S. Stefanucci. 2001. “Introducing workflow management in software maintenance processes”. In *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001,* 441–450. ISSN: 1063-6773. https://doi.org/10.1109/ICSM.2001.972757.

Bhatt, Pankaj, Williams K, Gautam Shroff, and Arun K. Misra. 2006. “Influencing factors in outsourced software maintenance”. *ACM SIGSOFT Software Engineering Notes* 31 (3): 1–6. ISSN: 0163-5948. https://doi.org/10.1145/1127878.1127883.

Bhatt, Pankaj, Gautam Shroff, C. Anantaram, and Arun Kumar Misra. 2006. “An influence model for factors in outsourced software maintenance”. *Journal of Software Maintenance and Evolution: Research and Practice* 18 (6): 385–423. ISSN: 1532-0618. https://doi.org/10.1002/smr.339.

Bhatt, Pankaj, Gautam Shroff, and Arun K. Misra. 2004. “Dynamics of software maintenance”. *ACM SIGSOFT Software Engineering Notes* 29 (5): 1–5. ISSN: 0163-5948. https://doi.org/10.1145/1022494.1022513.

Bianchi, A., D. Caivano, F. Lanubile, and G. Visaggio. 2001. “Evaluating software degradation through entropy”. In *Proceedings Seventh International Software Metrics Symposium,* 210–219. ISSN: 1530-1435. https://doi.org/10.1109/METRIC.2001.915530.

Bosch, Jan. 2009. “From software product lines to software ecosystems”. In *Proceedings of the 13th International Software Product Line Conference,* 111–119. SPLC ’09. USA: Carnegie Mellon University. Visited on June 11, 2022.

Bourque, Pierre, R. E Fairley, and IEEE. 2014. *SWEBOK: guide to the software engineering body of knowledge.* OCLC: 880350861. ISBN: 978-0-7695-5166-1.

Buchmann, Irene, Sebastian Frischbier, and Dieter Putz. 2011. “Towards an Estimation Model for Software Maintenance Costs”. In *2011 15th European Conference on Software Maintenance and Reengineering,* 313–316. ISSN: 1534-5351. https://doi.org/10.1109/CSMR.2011.45.

Cavalcanti, Yguarata Cerqueira, Eduardo Santana de Almeida, Carlos Eduardo Albuquerque da Cunha, Daniel Lucrédio, and Silvio Romero de Lemos Meira. 2010. “An Initial Study on the Bug Report Duplication Problem”. In *2010 14th European Conference on Software Maintenance and Reengineering,* 264–267. ISSN: 1534-5351. https://doi.org/10.1109/CSMR.2010.52.

Chapin, Ned, Joanne E. Hale, Khaled Md Khan, Juan F. Ramil, and Wui-Gee Tan. 2001. “Types of software evolution and software maintenance”. Num Pages: 28 Publisher: Wiley-Blackwell, 111 River Street Hoboken NJ 07030-5774 USA, *Journal of Software Maintenance and Evolution: Research and Practice* 13 (1): 3–30. ISSN: 1532-060X. https://doi.org/http://dx.doi.org/10.1002/smr.220.

Chen, Celia, Reem Alfayez, Kamonphop Srisopha, Barry Boehm, and Lin Shi. 2017. “Why Is It Important to Measure Maintainability and What Are the Best Ways to Do It?” In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C),* 377–378. https://doi.org/10.1109/ICSE-C.2017.75.

Chen, Celia, Reem Alfayez, Kamonphop Srisopha, Lin Shi, and Barry Boehm. 2016. “Evaluating Human-Assessed Software Maintainability Metrics”. In *Software Engineering and Methodology for Emerging Domains,* edited by Lu Zhang and Chang Xu, 120–132. Communications in Computer and Information Science. Singapore: Springer. ISBN: 978-981-10-3482-4. https://doi.org/10.1007/978-981-10-3482-4_9.

Choudhari, Jitender, and Ugrasen Suman. 2014. “Extended iterative maintenance life cycle using eXtreme programming”. *ACM SIGSOFT Software Engineering Notes* 39 (1): 1–12. ISSN: 0163-5948. https://doi.org/10.1145/2557833.2557845.

Desharnais, Jean-Marc, and Alain April. 2010. “Software maintenance productivity and maturity”. In *Proceedings of the 11th International Conference on Product Focused Software,* 121–125. PROFES ’10. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-0281-4. https://doi.org/10.1145/1961258.1961289.

Fitzgerald, Guy, Steve Counsell, and Jason Peters. 2013. “A Study of Web Maintenance in an Industrial Setting”. In *2013 17th European Conference on Software Maintenance and Reengineering,* 391–394. ISSN: 1534-5351. https://doi.org/10.1109/CSMR.2013.56.

Friedrich, Jan, and Klaus Bergner. 2011. “Formally founded, plan-based enactment of software development processes”. In *Proceedings of the 2011 International Conference on Software and Systems Process,* 199–203. ICSSP ’11. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-0730-7. https://doi.org/10.1145/1987875.1987908.

Gill, T. Grandon, and Alan R. Hevner. 2011. “A Fitness-Utility Model for Design Science Research”. In *Service-Oriented Perspectives in Design Science Research,* edited by Hemant Jain, Atish P. Sinha, and Padmal Vitharana, 237–252. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-642-20633-7. https://doi.org/10.1007/978-3-642-20633-7_17.

Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. “Design Science in Information Systems Research”. Publisher: Management Information Systems Research Center, University of Minnesota, *MIS Quarterly* 28 (1): 75–105. ISSN: 0276-7783. https://doi.org/10.2307/25148625.

Hira, Anandi, and Barry Boehm. 2016. “Using Software Non-Functional Assessment Process to Complement Function Points for Software Maintenance”. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement,* 1–6. ESEM ’16. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-4427-2. https://doi.org/10.1145/2961111.2962615.

———. 2018. “COSMIC Function Points Evaluation for Software Maintenance”. In *Proceedings of the 11th Innovations in Software Engineering Conference,* 1–11. ISEC ’18. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-6398-3. https://doi.org/10.1145/3172871.3172874.

IEEE. 2017. “ISO/IEC/IEEE International Standard - Systems and software engineering – Software life cycle processes”. Conference Name: ISO/IEC/IEEE 12207:2017(E) First edition 2017-11, *ISO/IEC/IEEE 12207:2017(E) First edition 2017-11,* 1–157. https://doi.org/10.1109/IEEESTD.2017.8100771.

———. 2022. “ISO/IEC/IEEE International Standard - Software engineering - Software life cycle processes - Maintenance”. Conference Name: ISO/IEC/IEEE 14764:2022(E), *ISO/IEC/IEEE 14764:2022(E),* 1–46. https://doi.org/10.1109/IEEESTD.2022.9690131.

Ivan, Ion, and Mihai Liviu Despa. 2016. “Estimating Maintenance Cost for Web Applications”. Num Pages: 10 Place: Bucharest, Romania Publisher: INFOREC Association, *Informatica Economica* 20 (4): 34–43. ISSN: 1453-1305. https://doi.org/10.12948/issn14531305/20.4.2016.06.

Kasanen, Eero. 1993. “The Constructive Approach in Management Accounting Research”, 23.

Krishnan, M. S., C. H. Kriebel, Sunder Kekre, and Tridas Mukhopadhyay. 2000. “An empirical analysis of productivity and quality in software products”. *Management Science* 46 (6): 745–759. ISSN: 00251909. https://www.proquest.com/abicomplete/docview/213173985/abstract/3BAEE5597CFF4AFBPQ/1.

Kumar, Sheo, Sugandha Chakraverti, S Agarwal, and Ashish Chakraverti. 2012. “Modified Cocomo Model For Maintenance cost Estimation of Real Time System Software”. *International Journal of Computer Science and Network* 1. Visited on August 21, 2022. https://www.researchgate.net/profile/Ashish-Chakraverti/publication/265186401_Modified_Cocomo_Model_For_Maintenance_cost_Estimation_of_Real_Time_System_Software/links/56c2068508ae44da37ff50e8/Modified-Cocomo-Model-For-Maintenance-cost-Estimation-of-Real-Time-System-Software.pdf.

Leotta, Maurizio, Filippo Ricca, Giuliano Antoniol, Vahid Garousi, Junji Zhi, and Guenther Ruhe. 2013. “A Pilot Experiment to Quantify the Effect of Documentation Accuracy on Maintenance Tasks”. In *2013 IEEE International Conference on Software Maintenance,* 428–431. ISSN: 1063-6773. https://doi.org/10.1109/ICSM.2013.64.

Lientz, B. P., and E. B. Swanson. 1980. “Software Maintenance Management”. *Addison Wesley, Reading MA.*

Martínez-fernández, Silverio, Claudia P. Ayala, Xavier Franch, Helena Martins Marques, and David Ameller. 2014. “Towards guidelines for building a business case and gathering evidence of software reference architectures in industry”. *Journal of Software Engineering Research and Development* 2 (1): 1–27. https://doi.org/10.1186/s40411-014-0007-5.

Mellegård, Niklas, Adry Ferwerda, Kenneth Lind, Rogardt Heldal, and Michel R. V. Chaudron. 2016. “Impact of Introducing Domain-Specific Modelling in Software Maintenance: An Industrial Case Study”. Conference Name: IEEE Transactions on Software Engineering, *IEEE Transactions on Software Engineering* 42 (3): 245–260. ISSN: 1939-3520. https://doi.org/10.1109/TSE.2015.2479221.

Nan, Ning, and Donald E. Harter. 2009. “Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort”. *IEEE Transactions on Software Engineering* 35 (5): 624–637. ISSN: 00985589. https://doi.org/10.1109/TSE.2009.18.

Napier, Nannette P., Lars Mathiassen, and Daniel Robey. 2011. “Building contextual ambidexterity in a software company to improve firm-level coordination”. *European Journal of Information Systems* 20 (6): 674–690. ISSN: 0960085X. https://doi.org/10.1057/ejis.2011.32.

Nguyen, Vu, Barry Boehm, and Phongphan Danphitsanuphan. 2009. “Assessing and Estimating Corrective, Enhancive, and Reductive Maintenance Tasks: A Controlled Experiment”. In *2009 16th Asia-Pacific Software Engineering Conference,* 381–388. ISSN: 1530-1362. https://doi.org/10.1109/APSEC.2009.49.

Niessink, Frank, and Hans van Vliet. 2000. “Software maintenance from a service perspective”. *Journal of Software Maintenance: Research and Practice* 12 (2): 103–120. ISSN: 1096-908X. https://doi.org/10.1002/(SICI)1096-908X(200003/04)12:2<103::AID-SMR205>3.0.CO;2-S.

Ooi, Ginny, and Christina Soh. 2003. “Developing an activity-based costing approach for system development and implementation”. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems* 34 (3): 54–71. ISSN: 0095-0033. https://doi.org/10.1145/937742.937748.

Parasuraman, A., Valarie A. Zeithaml, and Leonard L. Berry. 1985. “A Conceptual Model of Service Quality and Its Implications for Future Research”. Publisher: American Marketing Association, *Journal of Marketing* 49 (4): 41–50. ISSN: 00222429. https://doi.org/10.1177/002224298504900403.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. 2008. “A Design Science Research Methodology for Information Systems Research”. *Journal of Management Information Systems* 24 (3): 45–77. ISSN: 0742-1222. https://doi.org/10.2753/MIS0742-1222240302.

Penny, D.A. 2002. “An estimation-based management framework for enhancive maintenance in commercial software products”. In *International Conference on Software Maintenance, 2002. Proceedings.* 122–130. ISSN: 1063-6773. https://doi.org/10.1109/ICSM.2002.1167759.

Piirainen, Kalle A, and Rafael A Gonzalez. 2013. “Constructive Synergy in Design Science Research: A Comparative Analysis of Design Science Research and the Constructive Research Approach”, 29. Visited on August 14, 2022. https://www.researchgate.net/profile/Kalle-Piirainen/publication/262198751_Constructive_Synergy_in_Design_Science_Research_A_Comparative_Analysis_of_Design_Science_Research_and_the_Constructive_Research_Approach/links/02e7e539f4ccf19f88000000/Constructive-Synergy-in-Design-Science-Research-A-Comparative-Analysis-of-Design-Science-Research-and-the-Constructive-Research-Approach.pdf.

Rahman, Hanif Ur, Mushtaq Raza, Palwasha Afsar, and Habib Ullah Khan. 2021. “Empirical Investigation of Influencing Factors Regarding Offshore Outsourcing Decision of Application Maintenance”. Conference Name: IEEE Access, *IEEE Access* 9:58589–58608. ISSN: 2169-3536. https://doi.org/10.1109/ACCESS.2021.3073315.

Rahman, Hanif Ur, Mushtaq Raza, Palwasha Afsar, Habib Ullah Khan, and Shah Nazir. 2020. “Analyzing Factors That Influence Offshore Outsourcing Decision of Application Maintenance”. Conference Name: IEEE Access, *IEEE Access* 8:183913–183926. ISSN: 2169-3536. https://doi.org/10.1109/ACCESS.2020.3029501.

Rao, B.S., and N.L. Sarda. 2002. “Applicability of IEEE maintenance process for corrective maintenance outsourcing-an empirical study”. In *International Conference on Software Maintenance, 2002. Proceedings.* 74–83. ISSN: 1063-6773. https://doi.org/10.1109/ICSM.2002.1167754.

Ricca, Filippo, Maurizio Leotta, Gianna Reggio, Alessandro Tiso, Giovanna Guerrini, and Marco Torchiano. 2012. “Using UniMod for maintenance tasks: An experimental assessment in the context of model driven development”. In *2012 4th International Workshop on Modeling in Software Engineering (MISE),* 77–83. ISSN: 2156-7891. https://doi.org/10.1109/MISE.2012.6226018.

Ruhe, Melanie, Ross Jeffery, and Isabella Wieczorek. 2003. “Cost estimation for web applications”. In *Proceedings of the 25th International Conference on Software Engineering,* 285–294. ICSE ’03. USA: IEEE Computer Society. ISBN: 978-0-7695-1877-0, visited on June 11, 2022.

Sharon Christa, V. Madhusudhan, V. Suma, and Jawahar J. Rao. 2017. “Software Maintenance: From the Perspective of Effort and Cost Requirement”. In *Proceedings of the International Conference on Data Engineering and Communication Technology,* edited by Suresh Chandra Satapathy, Vikrant Bhateja, and Amit Joshi, 759–768. Singapore: Springer. ISBN: 978-981-10-1678-3. https://doi.org/10.1007/978-981-10-1678-3_73.

Shoukourian, Arsen, and Emma Danielyan. 2011. “How to measure “soft” things?” In *2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR),* 1–4. https://doi.org/10.1109/CEE-SECR.2011.6188466.

Sindhgatta, Renuka, Nanjangud C. Narendra, Bikram Sengupta, Karthik Visweswariah, and Arthur G. Ryman. 2010. “Timesheet assistant: mining and reporting developer effort”. In *Proceedings of the IEEE/ACM international conference on Automated software engineering,* 265–274. ASE ’10. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-0116-9. https://doi.org/10.1145/1858996.1859049.

Sneed, Harry M., and Wolfgang Prentner. 2016. “Analyzing Data on Software Evolution Processes”. In *2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA),* 1–10. https://doi.org/10.1109/IWSM-Mensura.2016.013.

Subramanyam, Ramanath, Fei Lee Weisstein, and M. S. Krishnan. 2010. “User participation in software development projects”. *Communications of the ACM* 53 (3): 137–141. ISSN: 0001-0782. https://doi.org/10.1145/1666420.1666455.

Yakura, Elaine K. 2001. “Billables: The valorization of time in consulting”. Num Pages: 20 Place: Thousand Oaks, United States Publisher: SAGE PUBLICATIONS, INC. *The American Behavioral Scientist* 44 (7): 1076–1095. ISSN: 00027642. https://doi.org/http://dx.doi.org/10.1177/00027640121956665.

Appendices

A Interview questions

General questions, arising from literature:

1. How do you feel about filling timesheets? What challenges do you face with time tracking?
2. Do you often work with multiple tasks in parallel, (like code review and learning activities along with development tasks, or several development tasks at the same time)?
3. Is it hard to deduce time spent for each activity afterwards?
4. We estimate tickets for customers. Sometimes the estimate isn't met and we have been working a lot shorter or longer with a ticket than estimated. Do you feel the need to fill in exactly the same number of hours than was estimated? Is there consequences for not doing so?
5. Do you think different categories of hours are valued differently?

Specific questions about the artifact:

1. Have you used the decision tree?
2. Do you find it useful?
3. What would make it better?
4. Did you attend the mapping exercise where we wrote our tasks or activities on each category we have for time tracking? Did you find that exercise helpful for determining which category suits a specific activity?
5. Have you noticed some task is missing from the instructions? Is there some tasks that you thought would be common but hasn't actually occurred at all in practice?
6. Has there been less need for you to fix your hours after you started using the instructions?
7. What do you think about our existing instructions?
8. Has the new instructions changed your feelings about time tracking?

Additional questions for persons responsible for checking the validity of hours:

1. How do you feel about your weekly task of checking reported hours?
2. Has transparency of sharing hours with an invoice to a customer benefited your relationship with the customer, or how customer perceives our efforts?
3. Have you used hour reports to spot trends - like workload balance, accuracy of estimation, effort per customer, and effort per service request type - and acted upon them?