Julia Sippola

Collaborative cognitive load in new junior team members in agile software development

Master's Thesis in Information Technology

September 5, 2022

University of Jyväskylä

Faculty of Information Technology

Author: Julia Sippola

Contact information: julia.t.k.sippola@student.juy.fi

Supervisor: Ville Isömöttönen, ville.isomottonen@jyu.fi

Title: Collaborative cognitive load in new junior team members in agile software development

Työn nimi: Yhteinen kognitiivinen kuorma ketterien sovelluskehitystiimien uusissa juniorijäsenissä

Project: Master's Thesis

Study line: Mathematical Information Technology

Page count: 68+2

Abstract: Software development is inherently a complex task, and starting in the field can be an overwhelming experience. With this semi-structured interview research study, we identify the themes affecting the collaborative cognitive load of new junior software development team members and present the results as thematic networks. Results revealed five main themes which affected the cognitive load experienced by junior developers. The themes were "problem-solving workflow", "searching for information", "group and teamwork related factors", "honing development skills" and "tools and instructions for newcomers". An additional sixth theme was also added, which described the perceived cognitive load, "descriptions of cognitive load". It could be argued that collaborative cognitive load did help lessen the cognitive load of the whole team after the introduction period for the newcomers, as the newcomers slowly became accustomed to the way of working and started contributing to the shared cognitive resource pool. However, some load factors did not decrese. For example, the expertise silos and silent knowledge were still seen as cognitive load factors, even after the introduction period was over.

Keywords: cognitive load, collaborative cognitive load, pro gradu -thesis, cognitive theory, agile software development, teamwork

Suomenkielinen tiivistelmä: Sovelluskehitys on kompleksinen tehtävä, ja alalla aloittaminen voi olla kuormittava kokemus. Tässä semi-strukturoidussa haastattelututkimuksessa tunnistetaan teemat, jotka vaikuttavat yhteiseen kognitiiviseen kuormaan uusien juniorijäsenten tullessa ketterään ohjelmistokehitystiimiin. Tulokset esitetään temaattisina verkostoina, ja kognitiiviseen kuormaan vaikuttavia pääteemoja löytyi viisi: “ongelmanratkaisun työnkulku”, “informaation etsiminen”, “ryhmä- ja tiimityöhön liittyvät kuormittavuustekijät”, “kehitystaitojen kehittäminen” sekä “ohjeet ja työkalut uusille”. Kuudes teema, “kognitiivisen kuorman kuvaajat” lisättiin omaksi teemakseen, ja siihen on koottu haastateltavien omat kuvaukset siitä, mikä on kuormittavaa. Tulosten perusteella näyttäisi siltä, että yhteinen kognitiivinen kuorma helpotti yksittäisten ihmisten kokemaa kognitiivista kuormaa perehdytysajan loputtua. Huomioitavaa kuitenkin on, että ainakin asiantuntijoiden siiloutuminen sekä hiljainen tieto aiheuttivat vastaajille kognitiivista kuormaa vielä perehdytysajan loputtuakin.

Avainsanat: kognitiivinen kuorma, yhteinen kognitiivinen kuorma, pro gradu -tutkielmat, kognitioteoria, sovelluskehitys, tiimityö

List of Figures

List of Tables

Contents

1 Introduction

Software engineers handle an ample amount of information daily while working. To successfully work in the field, the engineers need not only to possess high technical skills, but they also need good communication and collaboration skills (Uludag et al. 2018). Agile software development is the current standard in the industry (Hoda, Salleh, and Grundy 2018), and the agile methods aim to maximize customer value and quality with increased development and release speed while emphasizing the social side of development. The basis for the agile methods is the agile manifesto, which always favors the individuals and their behavior over non-changeable and strict processes (Beck et al. 2001). Thus, software development is a field filled with heaps of collaboration and teamwork.

Software development is essentially a complex task (Clarke, O'Connor, and Leavy 2016; Banker, Davis, and Slaughter 1998), and the average engineer has to consider multiple different requirements to provide well-functioning applications with few to no bugs. The requirements can come from many different sources: the customers, the architects, code quality, and maintenance requirements. The expectations for a new software developer are constantly changing as new technologies are developed, even if the basic idea of programming stays the same. On the other hand, experienced developers are also expected to know and learn about new technologies to produce high-quality, on-demand, reliable, and fail-safe products.

How does all this affect the new members of software development teams? This semi-structured interview research study tries to discover which themes and topics are thought to be drivers of cognitive load in new software development team members and which themes or topics may ease the load. This study will also explore the concept of collaborative cognitive load theory and apply that to the thoughts and themes arising from these interviews. Finally, the results of the analysis are shared as thematic networks in Chapter 3.

Why is this subject important and relevant? Collaborative cognitive load is a fairly new concept presented by Kirschner et al. (2018). Cognitive load theory itself was introduced back in the 1980s by John Sweller (1988), and it has been a widely accepted and researched

subject after that. Most of the studies are focused on school and learning environments — it is fascinating to learn more about them outside of those domains. Thus, the environment and context chosen for this particular study are that of a software development team, when a new team member is introduced to the group. A part of the motivation for this particular context was my own struggles, as I was starting my career as a software developer around the time of writing my thesis.

As cognitive load theory is principally focused on individuals, collaborative cognitive load theory presents the concept in a wider context. Janssen and Kirschner (2020) expand the theory to include multiple information processing systems, such as a team or a group. Kirschner, Paas, and Kirschner (2009) stated that as task complexity rises, learning is more effective and efficient when learning as a group of individuals rather than learning individually. Also, when task complexity is high, dividing the processing of information across individuals shares the cognitive capacity of a group (Kirschner, Paas, and Kirschner 2009), which increases the total amount of processing capacity. However, it should be noted that such divisions require the information to be recombined and reprocessed as it is shared between individuals, and thus it requires some cognitive effort as well (Kirschner et al. 2018). It also requires the learners to have unified schemas of the subject at hand (Janssen and Kirschner 2020).

The thesis will introduce the theoretical background of cognitive load theory and collaborative load theory in Chapter 2. The chosen method, data collection procedures and the analysis process of the interviews are explained in-depth in Chapter 3, and Chapter 4 showcases the results. Chapter 5 will offer the discussion and common themes and findings as well as some food for thought about the introduction period of newcomers. The conclusions are presented in Chapter 5.

2 Theory, Related Work and Background

This chapter introduces group processes, cognitive load theory and collaborative cognitive load theory. The chapter also includes simple demonstrations of these theories by applying them to software development.

2.1 Becoming a member of a team

Tuckman (1965) proposed group development process to have four stages. Later Tuckman and Jensen (1977) revised these stages and the four stages grew with one more stage. Those five stages are called forming, storming, norming, performing, and adjourning stages: these are all essential for a team to grow, learn, solve problems, and find solutions.

Tuckman and Jensen (1977) presented these stages as follows: the **forming phase** is the initial stage where the team is first put together. Team members may feel mixed and undecided, and conflict is avoided. Group members want acceptance from other members. Team members look to a group leader for guidance and advice. To advance, each member must accept the threat of conflict.

Storming stage begins as group members start to organize tasks and process surface-level conflicts. Leadership, power, and structural issues are apparent at this stage. To progress, the group must move on to a problem-solving mentality instead of fighting for power, and they have to start listening to each other.

In **norming stage**, team members create new ways to collaborate and develop cohesion. Leadership is not solely on one person; the group works as a team. Team members must learn to trust each other. Creativity is high, and the information is shared freely between team members. The drawback of this is that group members may start to resist change since they are comfortable with how things are right now.

Performing stage is the stage where the team is flexible, individuals are adaptive, and they meet the needs of other team members effectively. It is the stage of true interdependence. All groups may not reach this stage.

The added stage of **adjourning** comes into effect when team members are ready to exit and terminate the team. A significant team structure, membership, or purpose change may happen here. This stage disengages the individuals from their group and team relationships.

Software development is done within teams, and team and group dynamics are also necessary to consider. Rupert Brown (1988) has defined the group as "*two or more people possessing a common social identification and whose existence as a group is recognized by a third party.*"

In a group setting, it is expected that the minority will assimilate to the majority, and usually, it takes some time for the newcomer to adjust to the pace and rhythm of the team before finding their voice within the group (Levine, Choi, and Moreland 2003; Moreland and Levine 1989). Therefore, their status as "the newcomer" can stay with them for an undetermined amount of time, or in some cases, it can only be lifted when someone new joins the team again.

According to Berger, Cohen, and Zelditch (1972), the status difference between the task-oriented group determine the prestige and observable power of a specific team member. The status difference is affected by the external status characteristic[1], even if they are not directly related to the task. This indicates that the already-existing statuses for other members of the team can play a huge part in welcoming the newcomer to the team. Age seems to affect group perception as well, as Schloegel et al. (2018) shows in their empirical study: middle-aged employees were favored over younger and older employees in agile software development.

Studies in group behavior show that the formal status structures operate not only for their established purposes but also for totally non-relevant tasks (Kalkhoff and Barnum 2000). The established purposes are the activities that the group is created to perform, which is to develop software in this case. Non-relevant tasks are things that are not directly relevant to the given action, such as having lunch together.

It is equally important that the newcomer feels welcome within the team, but it is equally essential that the team members feel that the new addition is a part of them (Levine, Choi, and Moreland 2003). From a more psychological point of view, humans have such an overwhelming need to feel accepted and belong that our whole self-esteem may rely on it (Hogg

1. e.g., age, gender, occupation and education level

and Gaffney 2018).

Greer et al. (2018) stated that hierarchy decreased team effectiveness, which is affected by team structure, like skill differentiation and membership instability, and hierarchy mutability. This differentiation makes hierarchical teams prone to conflicts and reduces team effectiveness. Task ambiguity enhanced team hierarchy, and its effects (Greer et al. 2018). Relating to this, the complex systems in software development can have a lot of undocumented restrictions and curiosities, which can seem like ambiguities (Massey et al. 2014).

Social identity theory (Tajfel 1974) proposes that the groups to which people belong are an essential part of personal identity, and they give us a social identity, a place to belong. It is vital that we see ourselves in the in-group ("us") rather than the out-group ("them") since the group members of the in-group will eventually detect more negative features about the out-group. The other members of the in-group should consider us to belong in the in-group to be able to achieve a feeling of belonging in the same group (Tajfel 1974).

Status plays a part in defining if a person belongs in an out-group or in-group. Berger, Rosenholtz, and Zelditch (1980) explained that any process in a social setting, in which the visible characteristics of individuals become the justification of inequalities, is regarded as a status-organizing process.

A study by Kalkhoff and Barnum (2000) implicated that the social identity theory (Tajfel 1974) and status characteristics theory (Berger, Rosenholtz, and Zelditch 1980) are interrelated through their interactive effect on each other. Thus, we cannot remove the status from our social identity or vice versa.

2.2 Cognitive load theory

Cognitive load theory (later referred to as *CLT*) has traditionally focused on individuals (Kirschner et al. 2018). Historically, Miller's article back in 1956 presented the capacity of a person's memory to be the magical number seven (Miller 1956). Since then, multiple researchers have refined the theory and concluded the capacity to be chunks of data, not just specific items. Cognitive load theory was later refined and introduced by Sweller (1988),

and the cognitive load theory has been a subject for researchers ever since.

The more a person knows about the subject beforehand, the more items can be interlinked, and more can be memorized since the person already has some basis to build more knowledge (Sweller, Merrienboer, and Paas 1998). On the other hand, pure, non-interlinkable items, like a random sequence of numbers or words, are harder to link together and so remembering them needs more processing power from the brain, and the harder they are to store to memory (Sweller, Merrienboer, and Paas 1998).

Cognitive load theory focuses on the cognitive architecture and its information processing concepts (Sweller, Merrienboer, and Paas 1998). Recent studies about learning and instruction advise that the usage of the cognitive resources during the learning process should provide interesting viewpoints to human cognitive processes (Paas and Van Merriënboer 1993; Klepsch, Schmitz, and Seufert 2017). According to CLT, information is transferred from the working, short-term memory to the long-term memory by expanding and altering the already-learned schemas of the brain (Sweller 1988). CLT argues that when the limited capacity and processing power of an individual working memory is exceeded, learning becomes harder (Brünken, Seufert, and Paas 2010). Traditionally CLT has defined three independent sources of memory load, which are *intrinsic cognitive load*, *extraneous cognitive load* and *germane cognitive load* (Sweller, Merrienboer, and Paas 1998). These sources are illustrated in Figure 1. Even thought the different load types are presented as blocks with clear and defined lines, they can rarely be separated or distinguished from each other.

Kirschner, Sweller, and Clark (2006) argue that minimally guided or unguided instructions do not support the cognitive architecture of humans. They also point out that the evidence from empirical studies show minimally guided instructions to be inefficient compared to instructions which emphasize guidance and the learning process. Minimal guidance is efficient only after the learner has sufficient prior knowledge (Kirschner, Sweller, and Clark 2006).

2.2.1 Intrinsic cognitive load

Intrinsic cognitive load (ICL) comes from the inherent complexity of the task. Complex things are inherently harder to learn than simpler ones, and they have higher intrinsic cog-

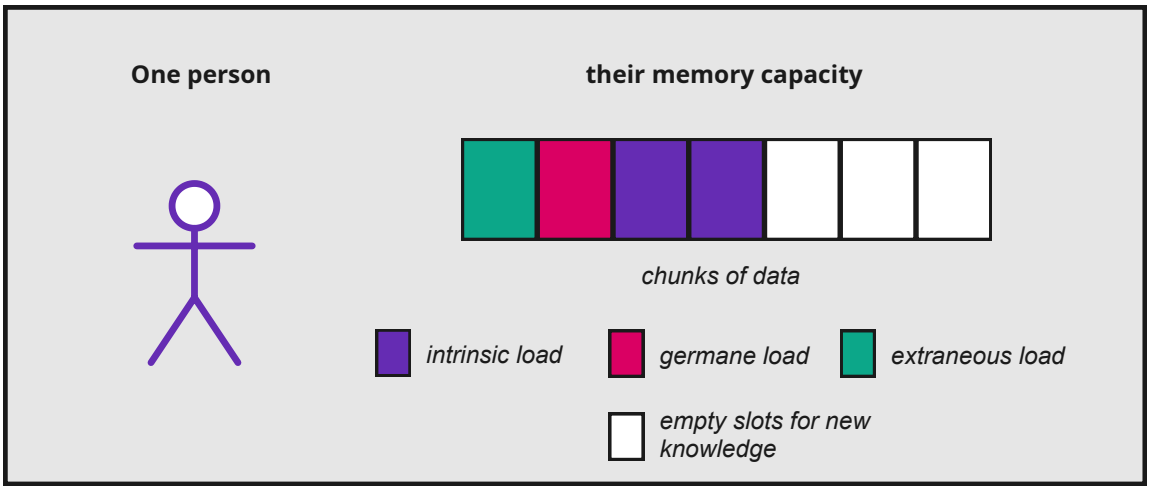

Figure 1: One person memory load example, illustrated by the author.

nitive load (Sweller 1994). There are two factors that affect the intrinsic load: (1) *element interactivity* and (2) *prior knowledge of the subject* at hand (Moreno and Park 2010).

The level of *(1) element interactivity* changes according to the total amount of different elements which the individual must process at the same time using the same working memory capacity and resources available (Chandler and Sweller 1996). Element interactivity can either be high or low. Low element interactivity means there is less elements that must be processed at the same time; those elements can be processed in sequences, they do not rely on each other, and there are minimal references to link them (Sweller 2010). High element interactivity implicates that the elements that must be processed are highly tied to each other and must be processed together - they are all taking space in the working memory simultaneously (Klepsch, Schmitz, and Seufert 2017; Chandler and Sweller 1996).

Matthews (2019) demonstrated low element interactivity and low intrinsic load for novice software developers. This example is elementary to understand the concept, and the actual, real-world example would be much more verbose and detailed. Figure 2 and Figure 3 shows two ways to implement a Django REST framework view. The code examples can be found from both Matthews (2019) and directly from Django Rest framework documentation[2]. The first one illustrates a more verbose way of writing and the second a more sophisticated so-

2. https://docs.djangoproject.com/en/4.0/

```
@api_view(['GET', 'POST'])
def snippet_list(request):
    """
    List all code snippets, or create a new snippet.
    """
    if request.method == 'GET':
        snippets = Snippet.objects.all()
        serializer = SnippetSerializer(snippets, many=True)
        return Response(serializer.data)

    elif request.method == 'POST':
        serializer = SnippetSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.
HTTP_201_CREATED)
        return Response(serializer.errors, status=status.
HTTP_400_BAD_REQUEST)

```

Figure 2: Code example taken from Django Rest Framework documentation: API view implementation example 1, low element interactivity

lution. Element interactivity is low in Figure 2 and very high Figure 3, resulting in bigger intrinsic load.

Following the Matthew's demonstration, Figure 2 shows one way to implement an API view with the Django framework. Even for novice software engineers, this code snippet would be pretty understandable when the reader has a basic understanding of HTTP principles and their usage. The code communicates what it is doing in every step: first, you see that this is something called "API view" that accepts `GET` and `POST` requests. The function takes in a "request" argument, just like you would assume an HTTP request would. Then, the top-level `IF` statement branches out to `GET` and `POST`, according to the given argument. `GET` gets all Snippet objects, serializes them, and then returns (presumably) the response containing all requested objects. `POST ELIF`-statement, in turn, creates a `POST` operation

```
    class SnippetList(generics.ListCreateAPIView):
        queryset = Snippet.objects.all()
        serializer_class = SnippetSerializer

```

Figure 3: Code example taken from Django Rest Framework documentation: API view implementation example 2, high element interactivity

of requested data, checks the validity of given data, and if true, saves it and returns `201 CREATED` response. If data is not valid, it returns `400 BAD REQUEST`.

The second example, Figure 3 contains all the same concepts as the first example, but it is much denser and packed with hidden information: the reader should already know a lot of concept this snippet utilises and a valid schema of how things work. As a result, there are fewer hints for the reader, and many questions are left unanswered. This is an example of high element interactivity in software engineering.

(2) Prior knowledge of the subject is the second factor that affects the intrinsic load. This is a critical building block for information processing and building: new information is linked with existing schema about the subject (Gerjets, Scheiter, and Catrambone 2004; Klepsch, Schmitz, and Seufert 2017).

The more structured and comprehensive the data were already in the learner's mind, the easier it is to link new information to them and add it to the prior schemata (Gerjets, Scheiter, and Catrambone 2004). On the other hand, too many unrelated elements during the presentation of processing of new information can be pretty distracting, and it takes space away from the working memory (Klepsch, Schmitz, and Seufert 2017).

Empirically, there has been efforts to try and reduce the intrinsic cognitive load by making the aforementioned two factors more manageable: *the segmenting principle* (a) to reduce the element interactivity and *the pretraining principle* (b) to reduce the intrinsic load caused by having insufficient prior knowledge (Klepsch, Schmitz, and Seufert 2017).

(a) *The segmenting principle* reduces the element interactivity by presenting the information

in smaller pieces, which helps in processing the information and links these bits of knowledge together (Mayer and R. E. Moreno 2010; Klepsch, Schmitz, and Seufert 2017). Each piece builds from the previous piece of information. Segmentation can be designed in many ways, and presenting the information in parts is usually helpful (Chung et al. 2015).

The segmenting principle is present in software development. Code formatting and stylizing usually helps us segment the pieces of code more efficiently (Wang, Pollock, and Vijay-Shanker 2011; Tashtoush et al. 2013). A more concrete example is provided by the author: if we compare Figure 4 and Figure 5, it should be evident that Figure 5 is easier and faster to read and decipher. Other examples would be the structure of classes and interfaces, understandable and human-readable parameters and variable names and clean project structures (Tashtoush et al. 2013).

```
    """
    This is a comment
    """
    def main():
        temp = input("Input the  temperature you'd like to convert?: ")
        degree = int(temp[:-1])
        input = temp[-1]

        if input.upper() == "F":
            result = int(round((degree - 32) * 5 / 9))
            output = "Celsius"
        elif input.upper() == "C":
            result = int(round((9 * degree) / 5 + 32))
            output = "Fahrenheit"
        else:
            print("Input proper value: C or F")
            quit()
        print("The temperature in", output, "is", result, "degrees.")


```

Figure 4: Code example without colors by the author

```
"""
This is a comment
"""
def main():
    temp = input("Input the  temperature you'd like to convert?: ")
    degree = int(temp[:-1])
    input = temp[-1]

    if input.upper() == "F":
        result = int(round((degree - 32) * 5 / 9))
        output = "Celsius"
    elif input.upper() == "C":
        result = int(round((9 * degree) / 5 + 32))
        output = "Fahrenheit"
    else:
        print("Input proper value: C or F")
        quit()
    print("The temperature in", output, "is", result, "degrees.")
```

Figure 5: Code example with colors by the author

(b) *The pretraining principle* reduces intrinsic load by relying on prior knowledge of the learner. If information about the subject is provided beforehand, we activate the prior memories linked with the knowledge the learner already has, thus increasing the integration of old and new information (Mayer and Pilegard 2005).

It should be noted that if we try to reduce the complexity of the subject that should be learned, it cannot be done without removing or altering essential information (Klepsch, Schmitz, and Seufert 2017). However, there have been efforts to lessen the intrinsic load of software development called cognitive-driven development (Tavares de Souza and Costa Pinto 2020), but the approach is still in the developmental stage. In this approach, the idea is to calculate complexity points for the source code, and keep the complexity in balance with the required quality.

2.2.2 Extraneous cognitive load

extraneous cognitive load (later called ECL) is regarded as the unnecessary cognitive load (Klepsch, Schmitz, and Seufert 2017). The extraneous cognitive load consumes the mental capacity of the learner for things that are not related to the learning itself, but which can be modified by removing the need for unnecessary processes. Thus, ECL is an unprofitable load that is affected by the instructional qualities of the task in progress (Sweller 2010).

Paas, Renkl, and Sweller (2003) supports this notion that extraneous cognitive load is unnecessary and loads the cognition without supporting the learning process. For example, instructional methods that requires learners to search for more instructions in the problem solution or explanation, or if the information is not readily available in the instructions, is expected to inflict a high extraneous cognitive load to the learner. This happens because cognitive capacity is consumed by processes that are unnecessary for schema acquisition or expansion (Paas, Renkl, and Sweller 2003).

The researchers have addressed how to reduce ECL in many different ways. Empirical studies suggest that so-called *multimedia principles* could help reduce ECL (Klepsch, Schmitz, and Seufert 2017). When the ECL is minimal as possible, it could free mental resources and more working memory capacity for in-depth learning (Klepsch, Schmitz, and Seufert 2017).

According to Mayer and Pilegard (2005), the essential multimedia principles are the segmenting principle, the pre-training principle and the modality principle. In essence, *the segmenting principle* means that the present learning material should be introduced in small, easy-to-handle bits (Mayer and Pilegard 2005). *Pre-training principle* means that the learner should be provided with information that is relevant and helpful in processing the learning material (Mayer and Pilegard 2005). *The modality principle* states that when information is presented in multiple ways (i.e., via a picture and a narration instead of a picture and text), the channels are not overloaded, and the learner retains more information from different sources (Mayer and R. Moreno 2003).

2.2.3 Germane cognitive load

Germane cognitive load (GCL) is the third type of cognitive load. GCL results from activities that facilitate learning to contribute knowledge transfer and form appropriate mental models and schemata, and it is considered to be beneficial for learning (Paas, Renkl, and Sweller 2003). Taking notes and explaining learned content to someone else are regarded as activities that contribute to learning (Klepsch, Schmitz, and Seufert 2017). Thus, a high germane load is an indication of high engagement in the task, and it directs their mental resources to the learning process (Paas, Renkl, and Sweller 2003). In software development, people usually prefer to use hands-on learning instead of learning just by the books (Cai and Guo 2019), and so their germane load is positively affected, and that adds to the load in general. Researchers have also addressed several design principles to foster this type of cognitive load, like self-explanation or explaining the learning materials to others (Klepsch, Schmitz, and Seufert 2017).

It should be noted that all the different types of cognitive load use the same working memory resources available. Therefore, all aspects of cognitive load should be considered when addressing the total load on mental processes. However, it is still debated if the different types of cognitive load are independent and if they can added to each other, accumulating together (Orru and Longo 2019). The load can also change type; the exact instructions can be related to extraneous load in one case, and then to germane load in another. For example, a new person may need a written guide on how to perform something, as well as a graphic beside it, but a more experienced person may not benefit from the graphic at all, and they could even be distracted by the unnecessary detail (Van Merrienboer and Sweller 2005). The Figures 1, 7 and 8 illustrate these chunks of data as separate blocks, but it should be kept in mind that the load factors cannot be distinguished so clearly from each other.

2.3 Effects of heavy cognitive load

As discussed above, high cognitive load is a sum of many factors. Galy, Cariou, and Mélan (2012) state that high cognitive load demands the learner to give more resources to process the given information to be able to reserve it for long-term memory effectively. Those ex-

tra resources which are sacrificed reduce the processing efficiency and performance (Galy, Cariou, and Mélan 2012). The study of Galy, Cariou, and Mélan (2012) also indicated that task difficulty, individual alertness level and time pressure overloaded the mental capacity of individuals.

Van Merriënboer, Kester, and Paas (2006) indicate that complex tasks require different instructional methods than simple and straightforward tasks. The article argues that methods and teaching instructions which induce high germane cognitive load would be beneficial for learning. However, the methods to achieve high germane load, such as high element interactivity and limited guidance can trigger more intrinsic load. It can be overwhelming for people who are yet to get accustomed to those methods. Decreasing intrinsic cognitive load could provide more capacity for germane cognitive load, but it can only be done by tampering with the element interactivity of the task (Van Merriënboer, Kester, and Paas 2006).

Fakhoury et al. (2018) noted that linguistic antipatterns increase the cognitive load of developers while reading the source code. Linguistic antipatterns were introduced by Arnaoudova et al. (2013), and they are represented by recurring, poor naming and commenting choices as well as lousy documentation and implementation in source code. Thus, the intrinsic load can be reduced as the novice developer learns the conventions.

Lin (2010) studied the negative effects of high cognitive load on job learninfg in Taiwanese information technology industry. Role ambiguity and role conflict were found to be the sources of cognitive load factors, and their loading effect increases when time pressure is present. Task complexity also had an increasing effect on role conflict, while trust and shared vision negatively influence role conflict. Lin's study (2010) suggests that it is the management team is responsible to actively decrease role conflicts and strengthen the trust between employees.

Helgesson et al. (2019) have identified cognitive load drivers in development teams. Their focus was *tool usage*, which formed the first cluster of cognitive load drivers. The cognitive load drivers were caused by intrinsic, delay, and interaction-related problems. Intrinsic problems manifested themselves as poor suitability and adaptability . Delay-related problems were all things that had an absence of response — such as system downtime or just

slow responses altogether. The third theme was interaction issues, where the system did not indicate how to accomplish specific tasks, and users had to guess how to use those systems. Second interesting cluster was *information quality*. Integrity, reliability, and information organization were also considered cognitive load drivers. The third cluster was *work process*, and the identified drivers were related to wasted efforts, lack of automation and oncoming ad hoc work, be it in the form of implementation of new tools or processes.

Helgesson, Appelquist, and Runeson (2021) studied the load drivers among novice software engineers. They identified merge operations and version control as the largest challenges that novice developers face, and they addressed the root causes to the nature of agile software development: working in parallel and iteratively with others leads to merge conflicts since no-one has a clear understanding what is happening, and dynamic design and vague requirements prevent the teams from gaining a cohesive goal. However, it was not the only issue that arose from their research data - lack of tool support and functionality, tool integration, and tool complexity were taxing on research subjects. The absence of communication and documentation were also contributing factors. In addition, some concerns about flow disruption and unwanted task switching were also brought up.

In summary, when group members get acquainted with each other and realise the knowledge and proficiency their fellow group members possess, the whole team can rely on each other (Janssen and Kirschner 2020) — one person does not have to remember everything, as the memory capacity is also shared. Moreover, as team members actively take part in transactive discussions and knowledge transfer within their team, they also facilitate effective and efficient learning processes (Janssen and Kirschner 2020). This also helps in building conforming schemas together, which takes us to our next topic, collaborative cognitive load theory in Section 2.4.

2.4 Collaborative Cognitive load theory

As cognitive load theory has been mainly considered as a set of something only related to individual learning, Kirschner et al. (2018) expanded the cognitive load theory to consider groups and teams too, as illustrated in Figure 6. They argue that the maximum cognitive

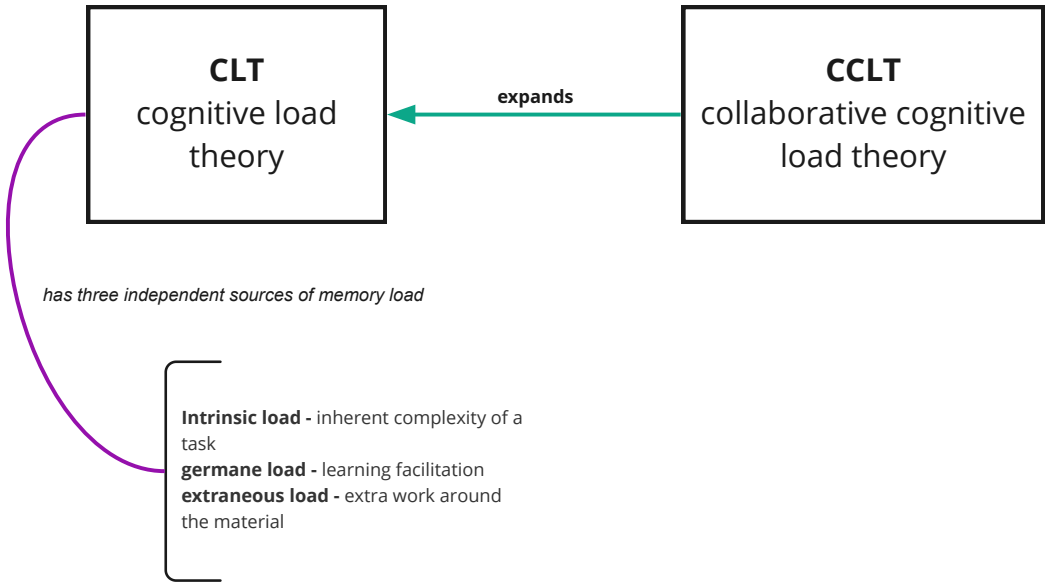

Figure 6: Collaborative cognitive load theory expands cognitive load theory, illustration by the author

capacity is not limited to the capacity of an individual, but that a group of individuals have a bigger shared cognitive space made up of the shared knowledge the team has. Therefore, collaborative cognitive load (later CCTL) considers a group of collaborative learners as information processing systems (Janssen and Kirschner 2020), thus achieving a bigger mental capacity. This is illustrated in Figure 7.

Janssen and Kirschner (2020) also bring collaborative cognitive load theory and collaborative learning together. In collaborative learning, to achieve the greater information processing capacity, the group members work together to learn assigned task or to reach a common goal (Johnson and Johnson 2009). The members should share the information and knowledge they have with each other during the task to reach the goal (Roschelle and Teasley 1995; Teasley and Roschelle 1993).

Chapter 2.2 describes how cognitive load theory studies the ways to optimize the cognitive load in individuals. It also tries to enhance the information processing capabilities and structures. In addition to that, collaborative cognitive theory highlights the positive interdependence between individuals in a group setting (Johnson and Johnson 2009): if there are

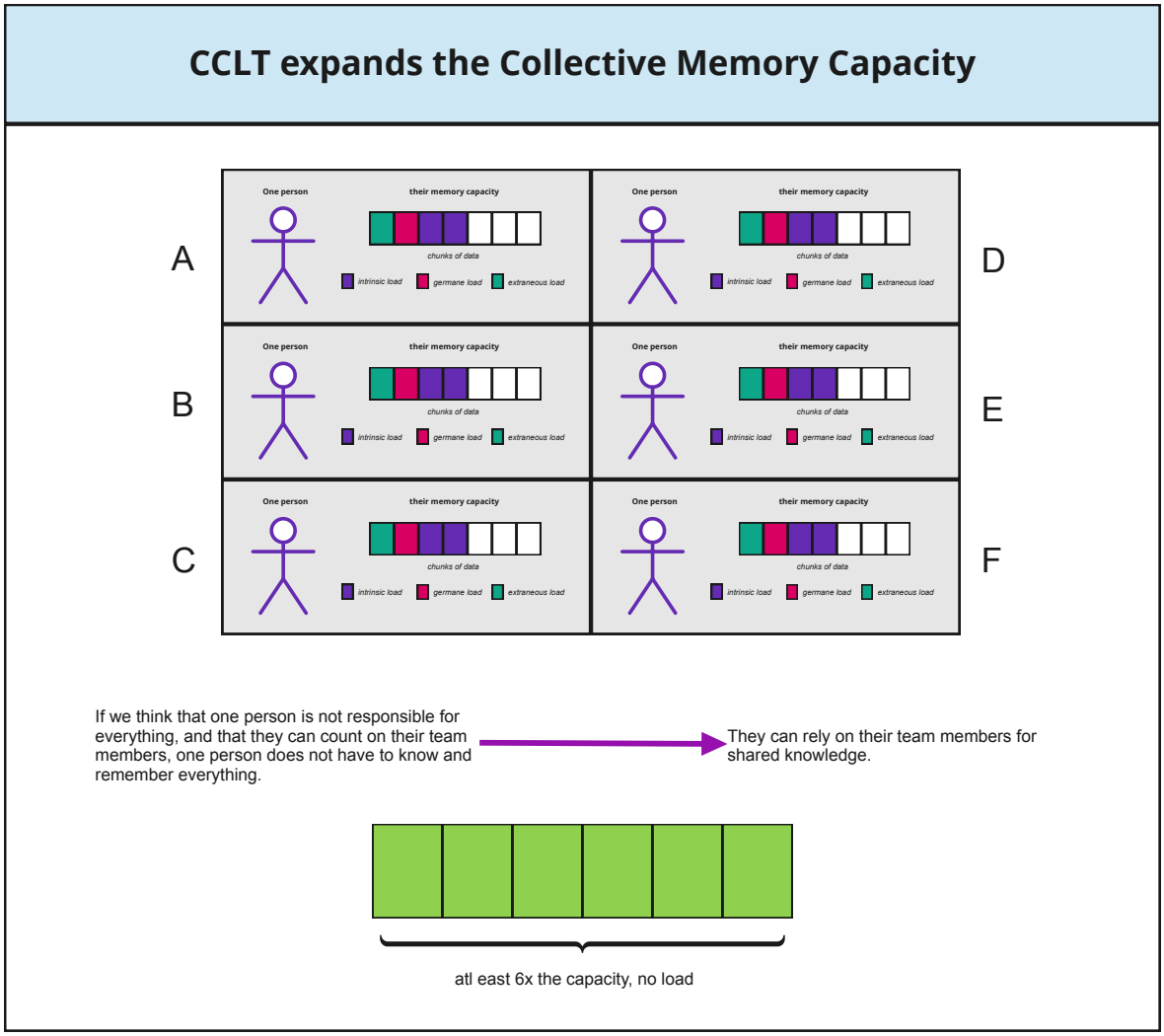

Figure 7: Whole team memory load example by the author

more people collaborating and sharing their working memory, the more information processing capabilities and cognitive resources they have available (Kirschner, Paas, and Kirschner 2009).

Kirschner, Paas, and Kirschner (2009) explain that in contrast to students studying by alone themselves, the collaborating group can use their shared cognitive resources to divide the intrinsic load of the task. A high element interactivity of the task is then shared over the available cognitive capacity and memory, decreasing the experienced cognitive load of an individual. Kirschner et al. (2018) defined this as follows: "*by sharing the burden of information processing with other group members, a collaborating student will need to devote*

fewer cognitive resources to seeking information and problem solving, compared to students studying individually" (Kirschner et al. 2018; Retnowati, Ayres, and Sweller 2017; Janssen and Kirschner 2020). This is demonstrated in Figure 7.

Later, Kirschner, Paas, and Kirschner (2011) showed that tasks that had a high cognitive load benefited more from collaborative learning, whereas low cognitive tasks were more efficiently learned by individuals. High cognitive load tasks required too many resources from individuals, exceeding their capacity, and processing the information was not as successful. Low cognitive load tasks did not require as much capacity, and collaborators and individual learners could process the information successfully, but collaborators had to use their resources to communicate and coordinate their efforts, which resulted in higher transaction costs. These transaction costs mean the cognitive and mental resources which are required from the collaborator when communicating and coordinating collaborative activities (Janssen and Kirschner 2020).

These transaction costs are disadvantageous to collaborating, as Janssen and Kirschner (2020) observed. According to Janssen and Kirschner (2020), members need different skills to achieve high-quality collaboration, and effective and clear processes that are known to all team members do provide a better basis for collaboration. However, individuals may experience the needed collaboration processes to be non-essential and extraneous to the their own work or learning, and some may even feel that the processes are completely unnecessary (Janssen and Kirschner 2020). According to an interview study by Stettina and Heijstek (2011), developers feel that documentation is important, but that the documentation in their projects is not available or that is not clear enough.

Figure 8 compiles the cognitive load theory and collaborative cognitive load theory visually together.

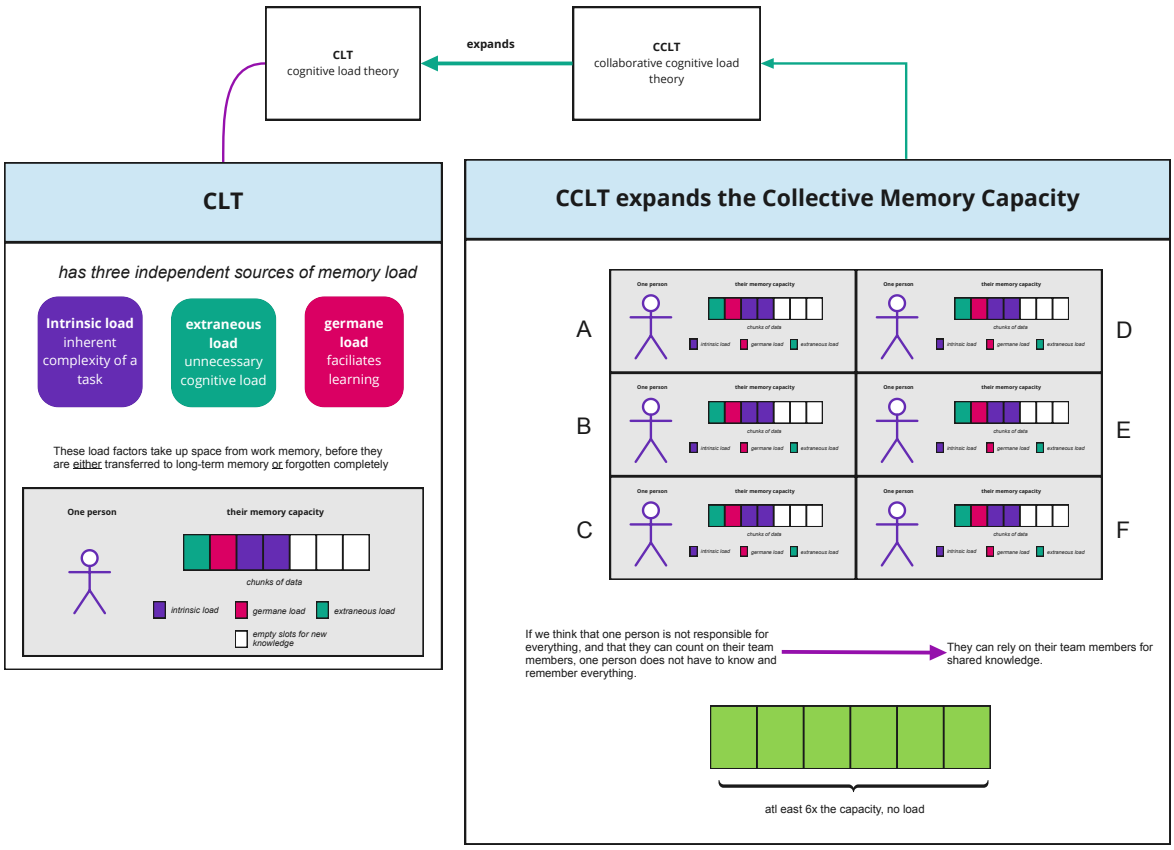

Figure 8: Assembled CCLT example by the author.

3 Research and Methods

This research study was conducted by interviewing six people who had recently joined an existing software development team. All the participants were employees of the same company, which is a traditional Finnish IT solution provider for the financial sector. Overall they have around 390 employees.

Interview questions were mainly derived from Chapter 2, with an addition of a new question regarding the perceived cognitive load after the first interview. The new question was added to get detailed and subjective descriptions of the cognitive load, as interviewees would describe what they felt as cognitive load. As a semi-structured interview, I had 20 leading questions that acted as a base for the interview, but interviews were allowed to stray from set questions as much as needed. The questions are attached to the end of this document.

3.1 Participants and data collection

The requirements for the interviewees were that they should have joined a new development team in recent months, maximum of 6 months ago, and that they should have less than four years of development experience. I had to expand the time frame for new team requirement from 6 months to 7 months, since some interviewees started working at the beginning of May 2021 as summer trainees, and these interviews were held from December 2021 to January 2022. Interviews were held in Finnish. Used quotations were translated to English by the author.

Team setup for the interviewees was from five members up to thirteen, with varying levels of seniority. All of these teams used scrum framework for agile development to some extent, for example every team held dailies but not all had a scrum master. Three of the interviewees were from the same team, and two interviewees had over two years of experience as developers, as can be seen in Table 1. Interviewee A and C had a little more work experience when compared to others. Interviewee C had changed teams three times prior to this interview, so they could compare the differences in cognitive load between teams.

Table 1: Table of interviewees and their experience and team setup

Identifier	Experience	Team size	Role	Been with current team
Interviewee A	4 years of work experience	five people	Developer	not specified
Interviewee B	6 months of work experience	ten people	Developer	6 months
Interviewee C	2.5 years of work experience	five people	Developer	4 months
Interviewee D	6 months of work experience	eight people	Developer / Tester	6 months
Interviewee E	7 months of work experience	five people	Developer	7 months
Interviewee F	1 month of work experience	thirteen people	Developer	1 month

The interviews were held as remote Zoom-meetings, which were recorded for transcribing. They lasted around 50 minutes per person. After the transcribes were done, the original recordings were destroyed. There were six interviews in total, from which the research material was formed.

3.2 Data coding

After the interviews were held and the author had transcribed them, interviews were categorized to initial low-level codes, which reflected the cluster of open-ended questions of the interview. First, the author used general codes to analyze the material. If the interviewee said something related to their daily working routines, that statement was put under the "teamwork" category, together with a note if the statement was said in a positive, somewhat negative, or neutral tone. After the first coding round, more defined codes were applied: "teamwork" would be the category, and it would have sub-codes like "teamwork: routines". This coding process followed loosely the guidelines of both Braun and Clarke (2006) and Jansen et al. (2010) on how to start and perform thematic analysis on qualitative data.

The author used Atlas.ti for the data coding and a digital notebook to refine themes and find connections between different codes. All together, 46 sub-codes and a total of 458 quotations were derived from the textual data.

3.3 Thematic network analysis

This research study analysis is based on thematic network analysis by Attride-Stirling (2001). These thematic networks are web-like structures with different abstraction levels. The levels are called basic themes, organizing themes, and global themes.

Themes were abstracted and identified after the coding described in Section 3.2. Refinement was done side-by-side as thematic networks were being constructed. After the initial coding, the author compared the codes from all interviews and analyzed statements under the same code. For example, "teamwork: routines" had several positive mentions of 'daily'-meetings, which the interviewees generally described as a place to bring up problems that they or their teammates had encountered while working on their tasks the day before. This finding would then be derived to "Asking assistance from teammates during scheduled meetings" as a *basic theme*, which is derived directly from textual data.

While analyzing the codes and creating basic themes, the author refined those basic themes to be more descriptive and "*(i) specific enough to be discrete (non-repetitive)*", and "*(ii) broad enough to encapsulate a set of ideas contained in numerous text segments*", following Attride-Stirling (2001) suggestion on how to create those basic themes. These themes are presented as the lowest level of abstraction in the thematic networks in Chapter 4.

After the creation of basic themes, *organizing themes* were created. Organizing themes are a middle theme: they group up more low-level basic themes and reveal more about the analyzed text, but the underlying concepts are more concrete here than in the global themes. Example of an organizing theme is "team routines ease communication barriers", as seen in Figure 11. All in all, 23 organizing themes were found. They are the middle level of abstraction presented in the network figures in Chapter 4.

Finally, the third level of themes is *global themes*. These global themes summarise the

organizing themes to tell us more about the text and what the themes tell us within the context. Our analysis found six global themes, which are *Solving problems*, *Searching for Information*, *Group and teamwork related load factors*, *Honing development skills*, *Tools and Instructions* and *Descriptions of Cognitive load*. They are presented in Chapter 4 as the highest-order abstraction levels at the center of the network figures.

4 Results

Six global themes were visible in the results, which affected the collaborative cognitive load of junior developers joining a new team. The sixth of these themes, *Descriptions of cognitive load*, gathered descriptions of load factors together. The five other themes were *Solving problems*, *Searching for Information*, *Group and teamwork related load factors*, *Honing development skills* and *Tools and Instructions*. This chapter describes the themes in detail.

The figures present the related global theme at the center. Organizing themes are one step lower, and basic themes are on the outer edges of the networks. Each subsection first goes through the high-level global theme and works its way downward. These figures can be zoomed in with ctrl++ if viewing the digital version of this document.

4.1 Problem-solving workflow

> *"First Google search, then plough through Verstas (company wiki) or ask from the team"* – Interviewee B.

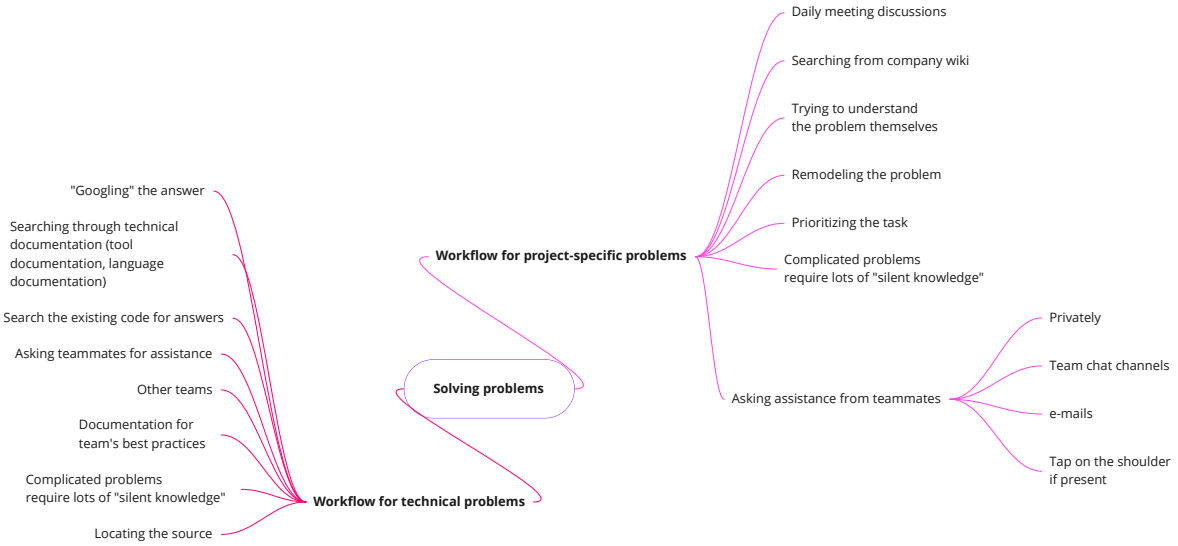

Figure 9: Problem-solving workflows

The global theme of "Solving problems" shows insights into how junior developers usually solve and divide their problem-solving workflows into two categories: *technical problems*

and *project-specific problems*. They have some overlapping in their basic themes, such as "Asking teammates for assistance ." The workflows are pictured in Figure 9. The division between questions depended on the type of question they had. If it was a technical question, like "how do I create a loop in this programming language," they would use search engines like Google to find examples from the internet. If, however, it was a project-specific question, like "what does this task mean" or "how do I create a situation like this in this program" they would either try to find them from the company wiki (Verstas / Confluence) and if that was unsuccessful, then they would ask their teammates. The division into two problem types is illustrated below:

> *"It depends on the problem if I first try to search Google some solution or examples on how to solve this thing, and if it is more application-specific then I might ask the team lead or someone."* – Interviewee E.

Most recognized ways to solve problems were using the Internet's vast knowledge pool to figure out possible solutions, searching documentation or company wiki to find answers, or asking for assistance from teammates. However, people did not state that asking their teammates was a first choice - they almost always preferred to first search for solutions on their own. Section 4.3 will describe this phenomenon in more detail, but junior developers seemed to experience some anxiousness and uneasiness when asking for help from more senior developers.

Even though asking for assistance was not the first option, interviewee F stated that asking teammates was usually the most effective and fastest way to figure something out. Interviewee D stated that they did not use the Internet for solving problems since their problems usually revolved around the product they were developing, and answers would generally not be found on the global Internet.

When asked what the ways to ask for assistance were, people mentioned the Daily, team-wide chat channels, e-mails, private chats, and taps on the shoulder if they happened to be working in the same location. Everyone mentioned the daily meeting. Daily is a scrum event, which takes place every day, and everyone in a team attends and tells what they did yesterday, what they will do today, and if they face any impediments in their work. Interviewees said

that these daily routines helped them immensely; it helped them focus and prioritize their work, they got to share their thoughts with others and ask if anyone could help them if needed, and they got to hear the progress of other team members' ongoing tasks.

Usually, at first you have one to two people close to you in the team who are your "go-to" people; if they do not know something, getting answers might slow down and get more challenging.

The Workflow for technical problems was fairly straightforward. Almost everyone mentioned that their first go-to tool was a search engine like Google or the technical documentation for the language they were programming or working with, as illustrated in the below quote. Searching the codebase for examples of other use cases was also used, and if those did not bear any fruit, they would go to the next daily or team channel to reach out to other team members.

> *"Hmm, it depends on the matter at hand. If I am just coding something and then start to think that maybe there would be some function which I could use or something, something that is related to details or to programming itself, then I would start searching the Internet, searching for information."* – Interviewee E.

More complicated technical problems usually require assistance from others. Some technical difficulties required new team members to contact other teams, which every interviewee saw as time-consuming process. Some problems required knowledge of things not documented by others or a much deeper understanding of the code that newcomers did not yet possess, leading newcomers to ask for assistance or help from other team members.

The Workflow for project-specific problems commenced with trying to understand the problem and remodeling it into something more familiar. The company wiki or a named mentor person was the first go-to place to look for specifications or explanations. As people were not that familiar yet when project-specific problems occurred, people would ask either the person who had introduced or assigned them the task regarding the said project. If that person were not available, they would go to the team's daily or team channel to ask for assistance from other team members.

Project-specific problems had the same notion as technical problems: a lot of information was not documented, and people had to reach out to more experienced people to get to the bottom of the problem. Interviewee A described this problem as follows:

> *"When talking about project-specific problems, information was not available since you usually had read the available documentation prior to this problem, and the problems presented in the documentation did not come up since you already knew the solution for it."* – Interviewee A.

4.2 Searching for information

> *"It's incredibly loading to explain the difficulty in a way which is understandable to someone other than my own rubber duck, to explain what the actual problem here is"* – Interviewee A.

Figure 10: Searching for information

"Searching for information" was identified as a global theme: it gathers statements that involve searching and finding relevant information together. Going from top to bottom, Organising themes were *No knowledge before hand*, *Ocean of information*, *Barriers in question formation* and *Hard-to-reach Information*. All of these made it harder for junior developers

to find correct and updated information that they could use. Figure 10 describes this global theme.

When discussing information availability, interviewees stated that it is hard for them to know what is essential at first glance. However, as time goes by, they start to recognize the crucial sections from documentation and the company wiki, which helps them focus their resources on the actual task instead of on every visible detail.

Hard-to-reach information refers to anything that is not accessible at the moment when searching for something, either because the documentation for it was lacking, missing, ambiguous, or in some other place that they had no access yet. Some people noted that the needed information might have been shown to them before, but they did not remember it at the time when they would have needed it. However, most interviewees noted that there was a vast amount of "silent" or "hidden" knowledge held by other team members that are not documented anywhere and that exists only in people's notes or minds but which is needed to proceed with a given task. An illustration is given below.

> *"It is a bit difficult when the information is not posted anywhere, or it is not visible, and then I have to remember it somehow, or I have to know who might know about this... I yearn for clearer instructions. "* – Interviewee E.

There was also talk about documentation - because the projects and systems people work with are complex and extensive entities, so is the documentation that describes those systems and their operations. Interviewee A noted that it takes some time to internalize and understand unfamiliar concepts when something new is laid out to learn without any easy grasping points, resulting in frustration.

As interviewee F states below, even if the documentation was available, it was not that trustworthy. If the team does not update the documentation as projects are finished or alterations are made, it gets outdated quickly, significantly complicating the already-complicated information availability. It also forced juniors to second-guess their findings and to confirm from someone if the documentation still applied.

> *"Maybe the most difficult thing was outdated documentation, I found the infor-*

mation I was looking for, but I still had to confirm from somebody if this still applies, is this still relevant, has this been updated, or is this outdated... So I received the information, but it was not trustworthy." – Interviewee F.

"I always had to check the year the documentation or specification was written in. There is lot of outdated information." – Interviewee F.

This **ocean of information** was also mentioned by all of the interviewees. There are some overlapping items with hard-to-reach information here since the amount of information is so massive that it is easy to lose oneself in the search. Most people mentioned that documentation and information are scattered around to so many different platforms that it is difficult to locate — possible searching places mentioned were company wiki, team-wide channels, meetings, emails, and private conversations. At times it was hard to know where to even to start the search. Interviewee B felt that the problem was the many possibilities where the information could be stored, as stated below.

"It is difficult because we have information about different automation and processes, everything work-related, so widely spread across different places. We have Verstas pages [company wiki], Confluence, and Teams-chats on our side as well as on the client's side, and then there are automation-specific chats that hold some information, and then we have some files in some Teams channels, and in some rare cases we even might have information on computer drives... The data is spread out on an extensive area." – Interviewee B.

No knowledge beforehand was expressed as an impediment. Interviewee B mentioned struggling with the unfamiliar terminology and that they did not yet know the environment, and that they did not know was the question that they had even relevant to their problem. Interviewee A stated that sometimes they did not know what to look for or how to start a task, as the example below describes.

"I had just started my first position as a developer, and I did not know how to use the terminology or the environment or anything that well that I could have described my problem shortly or efficiently, nor could I tell what I would have

wanted to do or what information I was looking for, it was not always easy." – Interviewee B.

Barriers in question formation: most significant effect for question formation was the team-level atmosphere. If the team-level atmosphere was welcoming, accepting, and encouraging, it was easy to ask questions and help. In addition, interviewee F noted that it was easier to ask from the Company wiki since it does not judge no matter how many times you ask — but team members might, especially if you did not know them that well yet. This effect was also present when newcomers had to contact other teams than their own.

Interviewee C felt that they did not understand the applications well enough, and that they would need someone with more experience to help even in forming the question. Interviewee C's statement is below.

> *"First, I would need someone with more experience to understand what I was trying to say and then they could help me to formulate the question."* – Interviewee C.

Interviewee A highlighted personality as a factor for how easy it is to ask for help or assistance — some people might feel anxious even admitting that they have questions regarding their work or that they do not know something. Help is hard to offer if others do not even realize they need assistance.

Compliments boosted the confidence and eased the self-doubt that newcomers felt. Support and availability were important for newcomers: if they felt they could trust and rely on their teammates, it lowered the threshold to ask for assistance.

Interviewees mentioned acceptable time to ask questions - if one is struggling with a task for a long time without asking for assistance, it is much harder to ask for help since you think others expect that you have made progress already. An illustration is given below.

> *"The more time it takes to form a question to progress even a little, the harder or bigger the threshold to ask it grows."* – Interviewee A.

Interviewees felt that a certain level of professionalism is required for asking questions and

that they first needed to understand what to ask clearly. Some interviewees felt their questions were too obvious or too easy for the more experienced people, and they should already know the answer. Some interviewees pointed out that asking the right questions required knowledge that they had not yet acquired.

4.3 Group and teamwork related factors

> *"As the work got more and more demanding, I felt nervous, I had to ask things from some strangers, and I was just a little developer-to-be, that was quite scary, and my questions must have been very obvious to them, but to me, they were not that obvious, so there was a threshold of some sort to ask those questions. "* – Interviewee C.

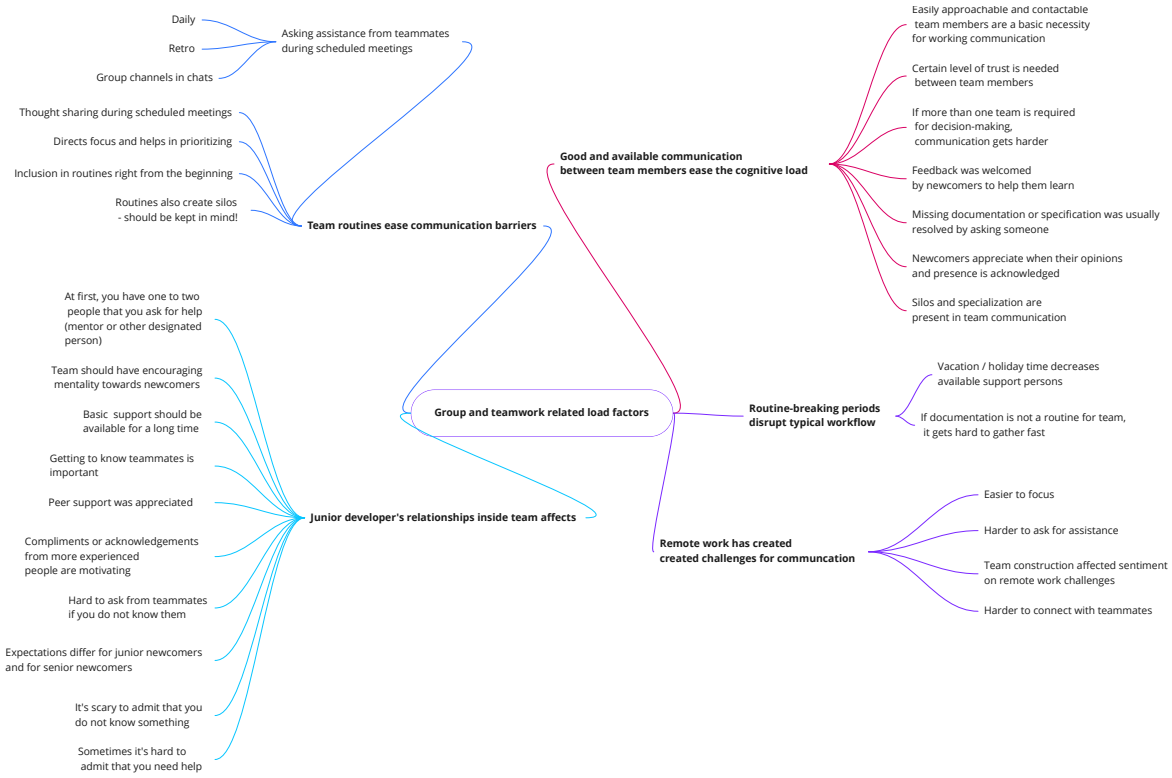

Figure 11: Group and teamwork related factors

Section 4.1 and Section 4.2 both mentioned team members as a crucial part of workflow. Solving problems and searching for information often required the newcomer to ask from other, more experienced team members. The global theme of "Group and teamwork related

factors" describes these factors in more detail.

The Global theme of Group and Teamwork related load factors is the biggest identified global theme. This includes five organizational themes: *Team routines ease communication barriers*, *Routine-breaking periods disrupt typical flow workflow*, *junior developer's relationships inside team*, *Remote work has created challenges for communication* and *Good and available communication between team members ease the cognitive load* . Refer to Figure 11 for the whole network.

Team routines ease communication barriers: Team routines like daily and retrospective were mentioned when talking about team communication and how information is shared. Retrospectives were easily accessible places to bring forth new suggestions on how to develop the way the team works: juniors felt that their suggestions and concerns were making an impact and helped the team grow. Daily was a place where the whole team was present, and people got to share their concerns and listen to others, which was a quick way to get an overview of ongoing matters. Dailies helped to prioritize and maintain focus on the crucial tasks. Dailies greatly supported the communication, as many interviewees stated. Examples below.

> *"[talking about daily] I could tell other team members what were my blockers and then we could discuss them together and ponder what we could do to those blockers as a group."* – Interviewee A.

> *"The atmosphere was good and I felt that I could ask and people responded and welcomed me with open arms.. They introduced and took me with them to their routines likes dailies and let me voice my thoughts even thought I did not have that much to say at the beginning."* – Interviewee E.

Routine-breaking periods disrupt typical flow workflow: Interviewee D noted that when people were on leave, for example, because of a holiday, it decreased the available support persons. It even made working very unpleasant since there was no one to ask for help or assistance from, and people just had to figure things out independently. Interviewee D felt that especially the summer season was hard, since more experienced people were out on vacations. Example below.

> *"I started as a summer trainee, so everyone disappeared after a few weeks and then suddenly you are working with only three other people, one of which was another summer trainee. So it was a bit hard to ask for help at that point."* – Interviewee D.

The routine breaks were also present when a team member was leaving the team, either because they switched jobs or changed their team. As a result, the people leaving would depart without having enough time to transfer their acquired knowledge to the newcomers.

Junior developer's relationships inside team affects cognitive load. This was visible in all interviews: people often stated that as a newcomer, it is hard to ask questions from others, especially if you do not know them. Mentorship and peer support were greatly appreciated, and most stated that at the beginning, they had one to two "trusted advisors", whom they contacted a lot and asked all the questions that came occurred to them. If they did not know the answer, they tried to provide juniors with the next person to ask or figure out whom to ask. An example is given below.

> *"There is one or two close individuals in that team and if they do not know the answer to something, for example how do I use email or something else, which is not their area of expertise, it makes the work slower and it forms its own bureaucratic blockers, which are not code dependant at all."* – Interviewee A.

Interviewees felt that introducing yourself to other team members was as important as getting to know them. This is illustrated below. As a new member got to know the others and as they heard a little bit about them and their personalities, it became easier to ask for assistance as well.

> *"I think it is important that you get to introduce yourself to the team, and of course, the team introduces themselves to you too that you get to know them."* – Interviewee C.

Multiple individuals noted that as the newcomer, they had more cognitive load to ask for help and might even be scared or afraid of their more experienced team member's attitude towards their questions. Some interviewees belittled their questions, saying that their questions must

seem so stupid, and they revealed that they do not know how to do this — this is why there is a threshold to asking questions from their seniors. Interviewees also stated that if they feel that their teammates are helpful and accepting, it takes much less effort to ask them work-related questions. Sometimes, if the teammates are deemed as scary, it can block new teammates from asking the question altogether. They may think that the attitude towards them is not welcoming or they are otherwise afraid of the reactions they will get. Even admitting that you do not know the answer to something can be a cognitive load factor for some interviewees.

Interviewee A noted that even though it is scary to admit that they do not know something and reveal to others that they do not know, they felt they had permission to be a beginner and ask those novice-grade questions. Interviewee A continued on that note and said that as time went by, their cognitive load increased since they assumed that other team members assumed that they had to know already, which created barriers to asking for assistance. Nevertheless, getting compliments from seniors motivated and reassured newcomers that they knew things and did a good job, as Interviewee D states below.

> *"Of course, as a newcomer when someone compliments your work or comments "good work" or anything, it really boosts the motivation to work."* – Interviewee D.

Remote work has created challenges for communication, as it is harder to ask for assistance, and one cannot just walk up to someone and ask their question directly. Interviewee A noted this in their quotation below.

> *"This.. antisocial behavior or threshold to contact others has increased since remote work came to be de facto."* – Interviewee A.

One interviewee stated that when you have to write your question, you must also think about your thought processes and assess if it is solid and understandable to others. The barriers in communication are here, too — again, if you do not know whom to ask or how to word your question out, even forming the question might be challenging. On the other hand, some interviewees noted that remote work lets them focus and concentrate much better without

all the extraneous background racket. At the same time, continuous notifications, remote meetings, calls, and pop-ups from instant messages and emails may be distracting when they flash on the same screen as you are trying to work on.

Good and available communication between team members ease the cognitive load: Certain level of trust is needed inside a team to communicate and collaborate, and easily approachable and contactable team members promote this. Almost all interviewees noted that getting help and assistance from their team was easy if they just asked.

Feedback was welcomed by newcomers as well, and feedback helped them learn. It also boosted motivation if it was a compliment about a well-done job. Newcomers also appreciated when others acknowledged their opinions and presence as full-fledged team members. Illustration of this given below.

> *"It is nice to work in this team, and I think it is important that my voice is heard too."* – interviewee C.

Some noted that teamwork did not work well when the matters needed other teams to collaborate with their team, and decision-making became much slower and ambiguous. Interviewees also noticed the existing silos and specialized knowledge inside their teams, but from their point of view, it was not necessarily a bad thing - just something that existed.

4.4 Honing development skills

> *"We have tried pair programming, which has been great especially when trying to develop new things in a project which requires a bit more investing and familiarization."* – interviewee A.

The global theme "Honing development skills" focuses on learning. As seen on Figure 12, *Learning helps to navigate the sea of uncertainty*, and *skillsharing* and *collaboration in development* facilitate learning and growing to be a professional developer.

Learning helps to navigate the sea of uncertainty and gain confidence in their work. As the newcomer learns the terminology and the standard practices of the team, their ability to

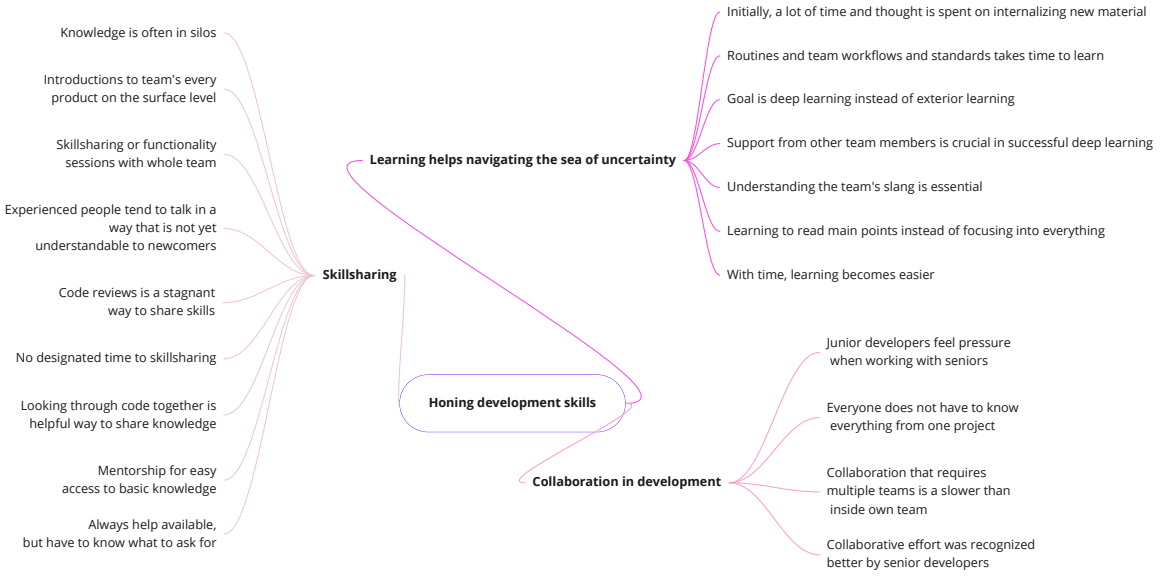

Figure 12: Honing development skills

find what is significant and what to focus on gets better. Interviewee A also describes the desire to understand the practices and underlying implications instead of just memorizing how to push the correct buttons.

In the beginning, support from others was crucial for learning. As time went on, junior developers learned whom to ask regarding different topics, and gradually they noticed that the more they knew, the less they naturally had to ask others. Interviewee E describes this in the example below.

> *"[when talking about documentation] well yes in theory documentation is pretty much the same things as before but I know more now and I'm not wondering and asking others all the time necessarily.. But still there are some details that I need to figure out and wonder how they were ought to be done"* – interviewee E.

Some also seemed a little frustrated because the learning material was so immaculate, and they could not remember everything immediately.

Skillsharing is important to junior developers. Interviewees A, C, and E stated that the knowledge is often in silos regarding specific projects or functions and that it would be good

to get knowledge about all ongoing products and projects that their team is working on and possible some visibility to other team's projects as well, especially if it affected their team in some way. Example of this silo effect below.

> *"The areas of expertise are so siloed, every younger member does that one thing that they have done since the beginning or starting in this team, and when we get a new ticket or request for new development or such, or to a specific development or product area, it has already been nametagged for that person who already knows about it. "* – interviewee A.

Interviewee A noted that more experienced people tend to talk in a way that is not yet understandable to newcomers, such as showing a technical explanation too fast or in a too high-level fashion. The quote below illustrates this.

> *"If the more experienced guy uses one IDE and shows us something, he has experience using that IDE and shows us how something is done programmatically, he might switch between screens and use his own shortcuts so fast that others do not necessarily understand what is happening."* – Interviewee A

Interviewee E noted that when juniors do code reviews, they sometimes only act as "stamp machines" since they do not yet understand the functionalities being implemented. Interviewee A mentioned mentorship and skill-sharing sessions as easy and productive ways to share the knowledge from team members with each other, as well as some other practices like pair programming.

Some interviewees noted that there was not really a designated time for skill-sharing or sessions with the designated mentor if someone was designated as one. They felt uneasy, as they thought the more senior members would be busy with other stuff.

Collaboration in development was recognized better by people with more experience. They often attributed working together as collaboration, while newer team members described this as getting help. Interviewees mentioned that they would at least like to know the basics from every project, since now, at times, they felt that if they do not know every detail of a project, reviewing merge requests was not a review, just a blind "OK" from someone else other than

the creator. Juniors felt pressure working with seniors: some said they felt they had to be better and should not ask "stupid" questions. Example below.

> *"I do not know that well what others are doing or working on since I have not familiarized myself with all the other projects that we have, so I do not really understand what they do... And then, for example, when I am doing code reviews, I do not really understand what is going on even if I am looking at their code. "*
> – Interviewee E.

Communication between different teams felt more complex than communication inside the team, creating delays in problem-solving.

4.5 Tools and instructions for newcomers

> *"[when talking about a technical tool] .. so if I had to do something that could be done in multiple different ways, I first had to figure out the good way of doing it to avoid doing it in a bad way, I had to confirm, verify and ask around about many customs and practices at the beginning."* – Interviewee B.

Global theme "Tools and instructions for newcomers" had four organizing themes: *Instruction quality*, *Stumbling rocks during introduction period*, *Tool usage sentiment* and *Prior knowledge of tools used differs from team standard*. Some tools were already familiar to new team members when they joined the team, but some were unfamiliar; those tools needed more time to adjust. Most of the cognitive load did not come from the actual usage of tools but from how the team used them and learning how to use them the same way.

Prior knowledge of tools used differs from team standard was described as the most striking when new team members came to the team fresh out of school or educational environment. One interviewee stated that their Git[1] practices in college did not make use of all the features available, and they only got to learn how to use Git properly at work. This is demonstrated with example below.

> *"In college, the projects are done alone, all by yourself, they are not group or*

1. Widely used tool for version control in software development, https://git-scm.com/

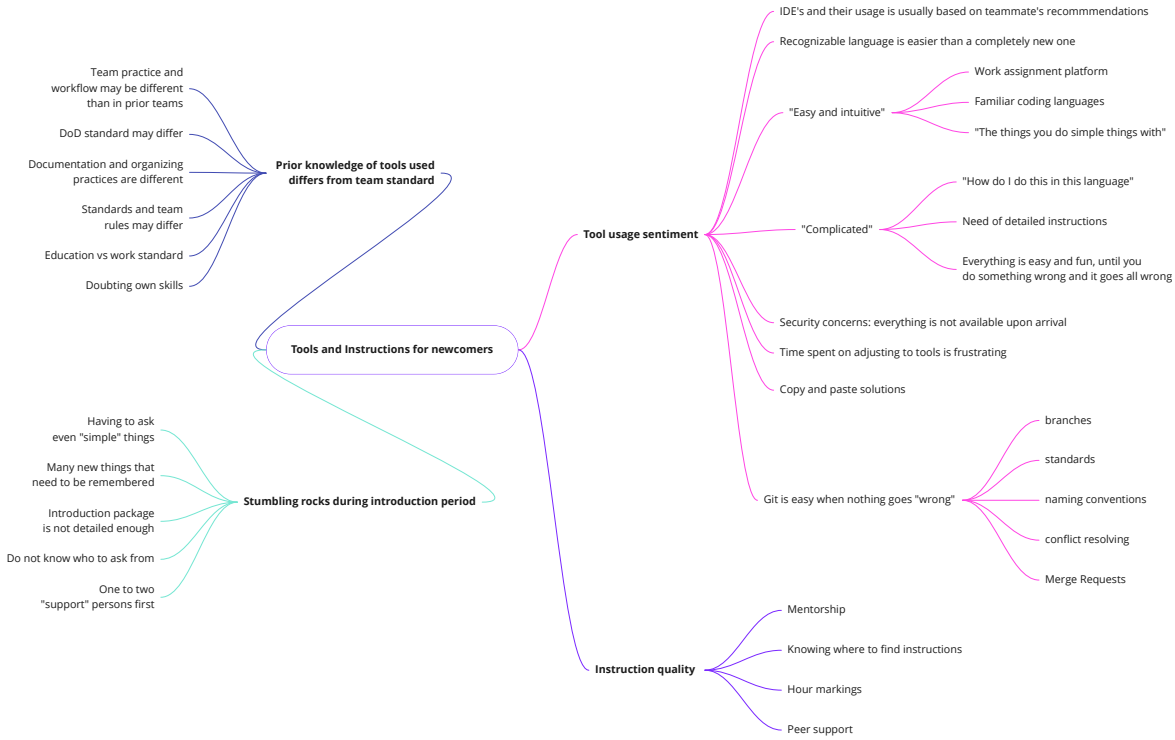

Figure 13: Tools and instructions for newcomers

> *team effort, and even if you do use Git, you do not have to think about merge conflicts or different branches or anything, since you know what you are doing and are the only one committing and you do not have to share the information to others."* – Interviewee F.

Team workflows differed — some teams followed SCRUM meticulously, while others may have had a more relaxed approach. As the approach varied, it took some time to get used to new standards and learn them, and juniors had to first figure out the standard practices of the team before doing anything else. The documentation practices were also different between teams, and as one interviewee had recently changed teams, they said that it took some time to get used to different ways of doing things. Some interviewees said that at the beginning, they doubted their skills, their knowledge of development, and do they have what it takes to be a developer who gets paid to program since their projects or school projects were so different from work.

Tool usage sentiment had a consensus that when you make no mistakes and use the tools

correctly, everything goes smoothly — the problems start when you make a mistake, or the tools do not work as expected. Again, a good example of this was Git usage — basic functions like "pull" and "push" were easy, but multiple branches, merge conflicts, and problems created while using them were usually harder to pinpoint and fix without help from other team members. Example given below.

> *"Git is a pretty good example of this. Everything is so easy when it is working, but the minute something goes wrong, I am just shrugging my arms, and I have no idea how to fix this"* – Interviewee A.

Work planning and assignment platforms, for example, VersionOne, were easy and intuitive. Interviewee E explained that tools like Jira were used for straightforward procedures like taking on assignments or marking a task done. More complicated things, which required multiple components and their usage at the same time or using produced software, were more challenging and required more specified knowledge since a simple push of a button was not enough to perform said task or assignment. Security concerns were also frustrating — tools did not work straight out of the box, and access to some services was restricted. In addition, to use them, you had to get approval and wait for some other team to operate, which could be time-consuming.

New team members also noted that they wished to learn how to do things on a deeper level, not only to "copy and paste" solutions without actually understanding them. Some noted that switching to a programming language they had no prior knowledge of also took some time to adjust to, but with enough time to learn the environment and the quirks of a language, it was not too great of a challenge.

Many people said that if there were options to what tools they used, like which IDE (integrated development environment) they wanted to use, they would usually go with one their colleagues already had. Changes were that if they got stuck with it, they would get help from others more effortlessly, and they would know how to help them then.

One interviewee also noted that since their team had had a big turnover just after they joined the team, they felt that the knowledge pool was diminishing faster than it was filled. When colleagues leave their current position or team or they exit altogether from the company, their

knowledge leaves with them.

Instruction quality refers to all available instructions, from development to hour markings or other administrative work. Interviewees wished for more detailed instructions for the basic tool usage. For example, how to run this program for the first time, run the tests, and install all needed certificates.

Mentorship and support from others were also regarded as a great thing. If the newcomer could not find the information they needed, they had to ask someone - and that someone was usually their mentor, if one existed, their scrum master or other teammates. Hour markings were mentioned a couple of times since it was not always obvious where they should mark their hours — interviewees wished for more precise instructions for this too. Interviewee C describes their introduction period below.

> *"[How did your teammates help you at the beginning, can you describe it?] They had planned beforehand that I needed support and had allocated a week for my orientation, and they had also named a mentor, a teammate, to plan what I should start with and what to learn. They initially gave me these simple tasks and helped me set up the environment, and the mentor was available when I requested help."* – Interviewee C.

Stumbling rocks during the introduction period were usually the sheer amount of new information that had to be memorized and connected in a short period — eventually, some details were forgotten. Feelings of frustration emerged fast if the information needed to proceed was unavailable, if the offered instructions did not work, or if there were no instructions at all and they did not have anyone to contact or ask help from. Sometimes interviewees felt that they had to be constantly asking even "simple" things that they thought everyone should know already, which was somewhat frustrating. Interviewee B describes their introduction page below, which was a good package.

> *"Someone had gathered an introduction package for me, which contained different matters that I should learn, and it came with links to relevant material in Verstas [the company wiki], and they had tagged a specified support or mentor there for me, who would teach me about the subject."* – Interviewee B.

4.6 Descriptions of cognitive load

> *"I think the Gaussian curve is quite illustrative. First, the load is not too bad; you can be, I mean, you can be merciful to yourself and grant yourself some concessions because you are the new guy on the team with new tools, and I can spend some extra time learning this and I can ask stupid questions since I am new. However, as time goes on and on, I start to feel that I should already know this and I should already be able to do this, and that is when my load grows bigger and bigger."* – Interviewee A.

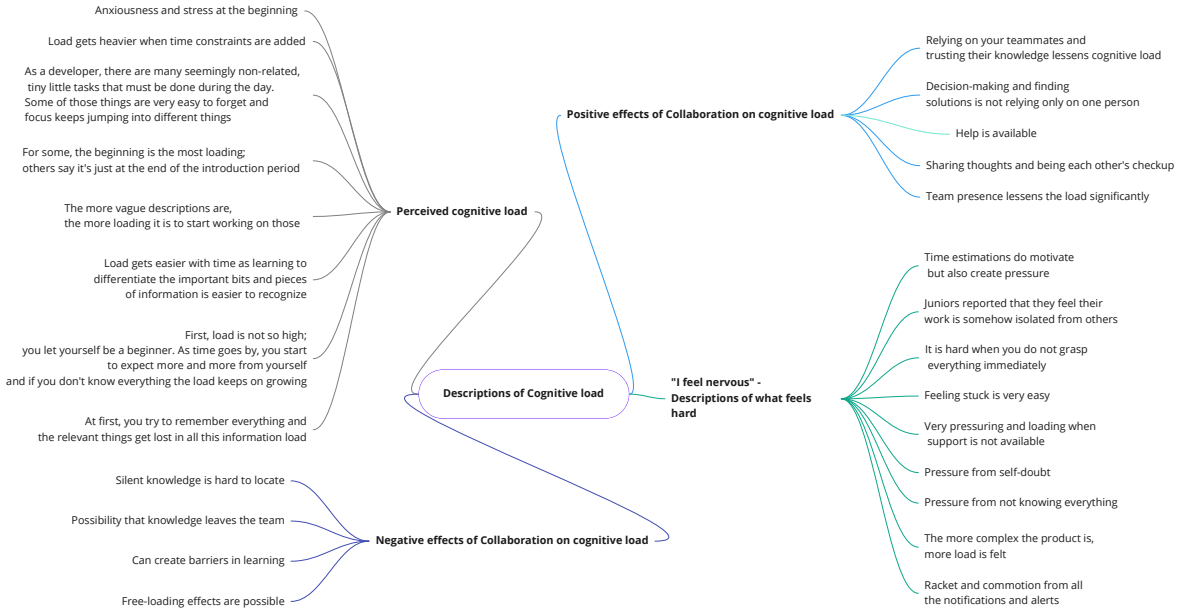

Figure 14: Descriptions of cognitive load

As this study wanted to focus on the collaborative cognitive load that new developers feel, I think it is justified that we include those descriptions here. The global theme "Descriptions of cognitive load" divided the descriptions into five different organizing themes: *Perceived cognitive load*, *Negative effects of collaboration on cognitive load*, *"I feel nervous" - Descriptions of what feels hard* and *Positive effects of collaboration on cognitive load*.

Positive effects of collaboration on cognitive load mentions that collaborative efforts like relying and trusting on teammates and their knowledge lessen the cognitive load. Decisions and solutions are not based solely on your own opinion, and others bring forth their viewpoints and considerations, opening up the possibility to expand and improve the solution.

Sharing thoughts and opinions was a good thing. The presence of a team was described to lessen the load since there would always be others working towards the same goal and guiding you on the way. Help is available. Illustrated below.

> *"[when asked if the support from team affected cognitive load] Yes, it did alleviate the load since I do not have to know everything by myself, and I can ask others if I do not know something, and only I do not make all the decisions; it is not solely my responsibility. And I can ask others too if I am creating a Pull Request, if this is ready, is this good enough, and when someone else double-checks it so no error of diligence or spelling mistakes or anything is left there, so that gives me a certain peace of mind that the code does not have to be perfect when it gets reviewed and that if there are mistakes I can still amend them before it is merged."* – Interviewee E.

Negative effects of collaboration on cognitive load stated that even though it is great that there are others, the silent knowledge is still hard to locate and dig out. Interviewee A specified that if the person possessing a vast pool of knowledge is not available, for example, leaves the workplace, then that knowledge is gone. This collaboration can also create barriers if the team atmosphere is not accepting or encouraging enough for the newcomers, so they do not dare to ask questions or guidance.

Interviewee D considered collaborative cognitive load from a team viewpoint — a significant amount of knowledge of an application's logic can be behind one team member, and other members know that if they have any questions about the functions or operational logic, they can ask that specific person. They commented that since one person is acknowledged as "the one", in a case something happens to them, there is no backup.

Descriptions on what feels hard are closely tied to self-doubt and self-negativity. Pressure from not knowing everything right away and getting stuck on something we are very loading, especially if there is no available help or assistance. Some juniors also reported that their work felt very isolated at times and was very independent. Interviewee B noted that all the notifications and alerts also raised their effective cognitive load, illustrated below.

> *"Sometimes there is so much background racket, and you start to think about all*

the notifications and the alerts and the commotion going on that how can you remember and focus on everything, I did notice that at times the cognitive load was quite big." – Interviewee B

People also noted that it was hard to give time estimations for how long a task would take them, since they did not know those complex systems that well yet, and some time would be needed to familiarize themselves with the material. More complex systems also required more time to learn; the more there is to learn, the more loading it felt. Time estimation that had to be done for work gave a feeling of anxiousness — what if I do not finish this or that in time?

Interviewee B stated that sometimes you may do many different things during your work day and that the focus shifts almost every hour, and you must constantly switch context during the day. Example given below.

"Regarding the cognitive load during my developer role, actually in previous work roles too, there is a lots of content switching during the day. The tasks that I'm supposed to do during the day can be very different and independent from each other and there is so much to remember in theory and in practice. Sometimes I get the feeling that my memory capacity is at its limit, will I remember to do everything during the day and if I forgot to do some minor things and I felt this especially at the beginning, when I was starting out." – Interviewee B

Perceived descriptions of cognitive load were dependent on the interviewee answering them — some felt that the load was highest at the beginning, and others stated that it was the highest when you know only some things but not all. However, all agreed that as you learn more and more, the load gets more manageable, and as you get to know your teammates, it is easier to ask and approach them.

Interviewee E felt that when they first started, the tasks given to them were too broad and it was difficult to locate the actual piece of code that should be altered to match the requirements of said task. Example given below.

"If the description of a ticket or a task which I am about to start is just a single

sentence, it is really far away from the actual implementation, and it is a much more complicated process in the code, and I start to think about all the borderline cases, which have not been thought beforehand and they make me wonder what is the desired outcome of this task. " – Interviewee E

5 Discussions

Inherently, software development is an excellent example of a complex task, especially when following the whole life cycle of an application from the beginning to the end of life (Clarke, O'Connor, and Leavy 2016; Banker, Davis, and Slaughter 1998).

The results of this study show that the most significant blocker seemed to be the vast pool of information offered, which often turned out to be outdated or hard to find. Eventually, things progressed only after contacting the correct person with the knowledge, but finding that person might be elusive since new team members are not yet familiar with whom to ask, whom to contact, or where to find that information. Asking questions from people that you do not personally know was considered exhausting and challenging: interviewees usually opted to try to figure things out themselves before asking for assistance.

5.1 How well did results resonate with theory?

As presented in Chapter 2, the cognitive load has three factors: intrinsic load, germane load, and extraneous load. These factors are hard to separate from each other, thus the perceived load is usually a mix of those three. The collaborative cognitive load theory of expanding cognitive memory capacity was observable after the newcomers had settled into their teams and overcame their initial information gathering phase, as they started to contribute their knowledge to the shared memory capacity.

The extraneous cognitive load seemed to be present as a load factor for newcomers. The results show that junior developers did not know where to start the search for information, and it depends a lot on the context of the information. This induces a high extraneous cognitive load, and should be avoided, as Paas, Renkl, and Sweller (2003) state. For example, project-specific information was more problematic to locate than purely technical information. As Sweller (1994) implies, the more established your route to information or possible solution is, the less extraneous load you have when pondering how to tackle complex problems.

The interviewees did not mention anything about code readability or code segmenting, which

should reduce the intrinsic load with the help of the segmenting principle (Mayer and R. E. Moreno 2010; Wang, Pollock, and Vijay-Shanker 2011). However, code segmentation is already embedded in the essence of programming in a way that it goes easily unnoticed when the programmer is accustomed to reading the source code. Thus, the help code segmenting provides in reducing cognitive load could be hard to perceive. The factors that help reduce the cognitive load can also be so unobtrusive that if one does not know how specifically to look for them, they can go incognito for a long time. The hidden clues which help us construe a our reality and understanding can be hard to register as they can happen unconsciously.

It was evident that many load factors related to other team members. Section 4.3 in Chapter 4 illustrated the problems that were present when you do not know your teammates. Relying on Tuckman and Jensen (1977), it seems that at least the newest member is still at the "forming stage" of group development stages. They want approval from other team members and look for guidance from them. Conflicts are avoided, and new team members try to learn how to work like others instead of questioning the current processes. As the newcomer learned new the ropes and started to trust other team members, they could speak up their mind more freely, which would indicate the approaching "storming" stage.

Continuing on the group development stages, some answers hint that the team was approaching the storming stage — like suggesting new ways of working or new practices — but downright conflicts between team members were not mentioned. There were no apparent signs of power struggles either. Instead, an invisible hierarchy was present: the people who had worked there for longer or had different titles were regarded mainly through their titles or seniority, as figures to look up to, not as equivalent team members by the newcomers. It should be noted that this view was that of the newcomers, and it would be interesting to hear the thoughts of the other team members with more seniority: would their opinions express the same forming stage, or would it be some other group development stage?

Even though collaborative cognitive load boasts of freeing up resources for extensive learning, juniors still need the basic knowledge and information to work efficiently. Accessible information is needed for this to work. If much time is spent just trying to figure out *where* something *might* be found, it loads the cognition and takes up resources from one person unnecessarily, resulting in a high extraneous cognitive load. Sweller (1988) suggests that

high load reduces the efficiency of learning. The unavailable information may be attributed to the transaction costs caused by collaboration (Janssen and Kirschner 2020): the costs at the time were too high for possible future collaboration to happen.

No knowledge beforehand was expressed as an impediment, as one could expect. It takes time to educate yourself about complex systems, and deciding how to proceed or start doing something might feel overwhelming when you feel like you do not have all the necessary information available to you. This is something the pre-training principle (Gerjets, Scheiter, and Catrambone 2004) of intrinsic load tries to lessen: adequate prior knowledge of the subject is essential, and without it, the intrinsic load increases.

As Helgesson et al. (2019) suggested, newcomers experienced Git and version control as a load factor, and they perceived the information structure to be inconvenient. As they learned more about their teams and the company's established best practices for Git usage and got help from others, the cognitive load vanished little by little. Git usage itself has high element interactivity, which affects the intrinsic load, especially if the schemas of the learner are not in sync with the actual flow of the tool.

The complex systems and their usage was also named as a load factor, but after learning the basics, those offered little to no worry: as the new team members learned and gained more experience, load factors regarding tool usage diminished. The schemas of the newcomers adapted to present the current system and it was easier to talk the same language as the more experiences team members.

Since newcomers trust that the more experienced people will know, sometimes it may create freeloading effects. The newcomers might think that since someone already knows this, they do not have to learn or remember it. However, those kinds of answers did not occur during these interviews.

5.2 Welcoming new software developers

Coming up with an appropriate question and finding the correct person to ask from is a challenge for newcomers. There are multiple factors: The vast pool of information is hard

to navigate, and knowing the right person to ask is also subjectively problematic. Knowing the right person is the key, which then again creates a loop of endless questions piling up on that specific person who is known to hold the needed information — thus, hidden knowledge starts to pool around specific people. This hidden knowledge can be hard to eliminate, but everyone should aim to minimize it. Team-wide discussions about upcoming solutions are a good idea to get the information flowing, and when not all teammates are present in a meeting, sharing notes of that meeting help in transferring knowledge. If there is a problem, possible solutions and workarounds should also be discussed publicly, preferably in writing, to show how the solution and conclusion was made. Then it would be easy to go back to those notes or texts and refer to them if something similar happens: then the solutions would not be hidden away in private chats or coffee table discussions.

One interviewee notes that when old teammates and colleagues switch or quit, they leave with tremendous knowledge. This supports the claim of a shared cognitive resources the collaborative cognitive load offer on a concrete level — the pool of knowledge has depleted with the exit of a team member. On the same note, the same interviewee also gives an example of how to lessen the impact of the withdrawal on the team: assignment instructions should be much more precise and elaborate, and the documentation from the start of the project until the end should be kept up-to-date. In addition, the individual who is leaving should share their current knowledge with the team before their departure.

Information and skill-sharing were things that newcomers wished for but which did not have designated time. The problem could be amended by designating the needed time, but if that is not possible, with clear documentation and instructions regarding different projects and products. The documentation should also be updated regularly — as the results show, even if there was documentation, it was not always trustworthy and needed to be double-checked by other members. Open and available communication and feedback inside the team also helps to decrease the cognitive load, as the results indicated.

When more experienced team members remember to document the thought processes of their solutions to problems or explain them to newcomers, they help the newcomers to form uniform schemas on solving those problems. It also helps in expanding the collective memory capacity, which is helpful in sharing the collaborative cognitive load. As Kirschner, Paas,

and Kirschner (2009) stated, the existence of conformed schemas between team members reduces the friction between the information that needs to be shared and distributed among the team members.

Even if documentation was deemed as one solution to the sharing of information, documentation in itself had problems that also needed resolving. Documentation was a separate task outside the actual development, and keeping the documentation up with development and vice versa was difficult. The interviews also pointed out that if the team did not update the documentation regularly, it got outdated quickly, significantly complicating the already-problematic information availability.

Getting information outside of own team was also considered a hindrance. The same effects were visible when asking their team members who were yet not familiar — newcomers had barriers in question formation. Questions like "what if I sound stupid" or "what if I am asking the wrong questions" were brought up in both cases. Remedies could include activities that foster the self-dependency and self-confidence in newcomers, but also setting up gatherings to give people a chance to get to know each other.

The newcomers should also be active in this regard, and have the courage to ask even the silliest of questions. The courage to ask was a much-talked insight during the interviews, and it seems that newcomers do not exactly fear to ask, but there is a threshold that needs to be overcome first. A designated mentor who is their guide eases the threshold. If a person is named a mentor, the mentor is a newcomer's first go-to person to ask — which is a great practice.

However, as the interviewees talked about their mentors, they usually mentioned things relating to the work. Therefore, I argue that the concept of a mentor should be extended to someone who promotes the social relationships between the team and the newcomer, as they already know the other members.

It was also notable that most of these factors lessened with time. The results indicated that newcomers acquired the needed skills to debug and work with scarce information to solve their problems as they gained experience. And, as their confidence increased, they dared to try their solutions before asking the seniors — and soon, they were the ones to teach the new

juniors about ways of working and how to find information.

5.3 Limitations and challenges of the study

During the interviews, it was evident that the questions leaned significantly on individually experienced cognitive load, and thus the answers focused on individual experience. For future work, I would recommend focusing on one specific team and conducting the interview within that team. During this study, all interviewed individuals had very different team constructions, and the experiences could vary significantly between teams. Thus, connecting answers entirely to collaborative cognitive load theory is challenging. For example, how does a new team member affect the cognitive load of the whole team instead of only individuals? Furthermore, how do we measure the cognitive load of a team accurately? Does every team member add to the shared cognitive resource pool, or do others decrease it?

Ayres (2020) and Lee et al. (2020) state that pupillometry was more reliable in accurately assessing cognitive load than subjective assessments like interviews. Accurate measures of cognitive load are critical in finding the clusters loading the cognition, but the most accurate measurements are still debated. Pupillometry also bypasses the subjective experience of cognitive load, and pupillometry needs highly specific research tools to capture the eye movements correctly - it might be hard to arrange that in a real-life location, such as a workplace. It could be possible to combine, for example, these two research methods and do a mixed-method research. However, the technical and physical arrangements for that present the same difficulties as pure pupillometry research, and getting the environment to be authentic enough would also be a challenge.

5.4 Trustworthiness

Nowell et al. (2017) and Braun and Clarke (2006) presents thematic analysis to be highly flexible approach in conducting data analysis on textual data. Thematic analysis can answer the need for detailed and complex data, even when the researcher does not necessarily have much experience in qualitative research approaches. Thus, it was a well-fitted approach for this thesis as a research method.

The disadvantages to this approach, however, are also the downfalls of a fresh researcher — the lack of precise processes and the freedom to choose how to conduct and present the analysis may result in a more incoherent presentation of the research as opposed to different qualitative research methods (Nowell et al. 2017). As this was the first thematic analysis that the author has done, they experienced the freedom of choice to difficult. The conventions of research were not familiar yet, and wondering how analyse and present the results took a lot of time.

The author has the power and the responsibility to present their findings reliably: hence, we shall explore the trustworthiness of this particular research study further. Lincoln and Guba (1985) added more dimensions to the trustworthiness than just validity and reliability. They introduced the notions of credibility, transferability, dependability, and confirmability. We shall explore these four areas in the light of this research and its trustworthiness.

Credibility should address the fit between the respondents', or in this case, the interviewees, views and the researcher's interpretation of them (Nowell et al. 2017; Guba, Lincoln, et al. 1989). For this study, the interviews with an individual lasted around 50 minutes, promoting interviewee engagement. The method was also discussed with peers and the master thesis' director to see if the chosen method was fitting. As the idea of the research was to search for factors that affect the cognitive load of newcomers, interview study was very fitting. This research study gives insight about the experienced cognitive load of newcomers. In the future the focus could be more shifted towards the collaborative cognitive load, and the interviews could be held between one specific team to observe the experienced cognitive load of the whole team.

The interviewees and the interviewer worked at the same company, so both parties were at leat somewhat familiar with each other. A trust bond was already formed between before the actual interview, so the interviewees did not have any reason to embellish their descriptions. The interviewees did not receive any debriefing of the initial results, nor were there any other data analyzers than the author — which can lead to only one interpretation of a statement or sentiment. However, the results did carry some of the findings other researchers have found.

Transferability accounts for the possible generalization of the question. In qualitative re-

search, the possible areas for transferability are impossible to pinpoint, so it is only up to them to provide descriptions of conducted research. This way, the generalizations can be transferred if the seeker of transferability so decides. This thesis provides insight into how the research was conducted and analyzed, so the transferability is up to the one seeking to transfer it (Nowell et al. 2017; Lincoln and Guba 1985). It should be noted that transferring the research study to any other company may produce different results — their newcomers may experience different themes of cognitive load, and their company culture and ways of working could be different.

Dependability is achieved by establishing the research progress in a clear manner, meaning that the process should be documented in a coherent, logical, and traceable way (Nowell et al. 2017; Tobin and Begley 2004). The author notes that, for the most part, the process is well-documented, logical, and traceable, but the analysis and making of the thematic networks could be more precise and methodological. The author did use a tool for creating the networks, which increases the traceability and documentation process, but having more than one researcher could have expanded the vision and it would not have depended so much on one interpretation.

Confirmability can be reached by distinctly identifying and indicating the interpretations and findings of the researcher from the data. This requires the conclusions to be demonstrated and illustrated clearly, and it should also explain how the conclusions were reached (Nowell et al. 2017; Tobin and Begley 2004). As the themes are derived from the answers provided by interviewees and examples illustrate them, there is clear evidence of how the conclusions were reached. Also, according to Guba, Lincoln, et al. (1989) confirmability is the sum of credibility, transferability, and dependability.

6 Conclusions

Five primary themes for collaborative cognitive load were identified: problem-solving workflow, searching for information, group and teamwork-related factors, honing development skills and tools and instructions for newcomers. Another theme was descriptions of cognitive load, which tied together perceived load factors.

Reportedly, load factors like tool usage and self-doubt about one's skills disappeared after the initial introduction period and were no longer consuming resources from learning. After a while, newcomers knew whom to ask and where to search for information about different problems, reducing cognitive load, arguably extraneous cognitive load. In conclusion, it could be argued that collaborative cognitive load did help lessen the cognitive load of the whole team after the introduction period, as newcomers slowly became accustomed to the team's way of working and started contributing to the shared cognitive resource pool. The benefits gained from transaction costs of collaboration overshadowed the temporary loss.

Software development requires extensive technical expertise as well as the ability to collaborate, communicate and work with others to build complex systems. Information availability and a well-working teams and individuals are all crucial for successful collaboration, but is is also essential that the documentation is up-to-date, available and the workflows, solutions and ways of working are clear to every team members.

Another conclusion from the results was that the expertise silos and silent knowledge held their place in being cognitive load factors well after the initial introduction period for the new team members was over. Future research agendas could be on how to prevent the silo effect and harness the collaborative cognitive load to aid teams and organizations to success. The transaction costs of collaboration in regards to cognitive load would also be intriguing to explore.

Further research of the collaborative cognitive load is recommended. This study tied collaboration loosely to the cognitive load perceived by new team members, and more knowledge about intra-team load would be needed. As collaborative cognitive load has been studied mainly in the field of computer-aided learning and education, more research on the field of

software development would give us better insights inside software development teams or even on a larger scale, like software development companies or the IT field as a whole.

Additional research on the cognitive load on more senior members would also be interesting. How is the cognitive load affected when a new member enters the team? Does it affect the cognitive load of a single person at all? Is the collaborative cognitive load affected? The senior members of the team possess the most domain knowledge of the product, and arguably the newer members rely on that knowledge a lot. It would be fascinating to see how the load is affected from that perspective, or is it affected at all.

Bibliography

Arnaoudova, Venera, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. 2013. ”A New Family of Software Anti-patterns: Linguistic Anti-patterns”. In *2013 17th European Conference on Software Maintenance and Reengineering,* 187–196. https://doi.org/10.1109/CSMR.2013.28.

Attride-Stirling, Jennifer. 2001. ”Thematic networks: an analytic tool for qualitative research”. *Qualitative Research* 1 (3): 385–405. https://doi.org/10.1177/146879410100100307.

Ayres, Paul. 2020. ”Something old, something new from cognitive load theory”. *Computers in Human Behavior* 113:106503. ISSN: 0747-5632. https://doi.org/https://doi.org/10.1016/j.chb.2020.106503.

Banker, Rajiv D, Gordon B Davis, and Sandra A Slaughter. 1998. ”Software development practices, software complexity, and software maintenance performance: A field study”. *Management science* 44 (4): 433–450. https://doi.org/10.1287/MNSC.44.4.433.

Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, et al. 2001. *Manifesto for Agile Software Development.* http://www.agilemanifesto.org/.

Berger, Joseph, Bernard P. Cohen, and Morris Zelditch. 1972. ”Status Characteristics and Social Interaction”. *American Sociological Review* 37 (3): 241–255. ISSN: 00031224. https://doi.org/10.2307/2093465.

Berger, Joseph, Susan J Rosenholtz, and Morris Zelditch. 1980. ”Status organizing processes”. *Annual review of sociology,* 479–508. https://www.jstor.org/stable/2946017.

Braun, Virginia, and Victoria Clarke. 2006. ”Using thematic analysis in psychology”. *Qualitative Research in Psychology* 3 (2): 77–101. https://doi.org/10.1191/1478088706qp063oa.

Brown, Rupert. 1988. *Group processes : dynamics within and between groups.* Edited by Rupert Brown. Lisäpainokset: Repr. 1989. - Repr. 1992. - Repr. 1993. - Repr. 1994. - Repr. 1995. - Repr. 1997. Oxford: Blackwell.

Brünken, Roland, Tina Seufert, and Fred Paas. 2010. ”Measuring Cognitive Load”. In *Cognitive Load Theory,* edited by Jan L. Plass, Roxana Moreno, and RolandEditors Brünken, 181–202. Cambridge University Press. https://doi.org/10.1017/CBO9780511844744.011.

Cai, Carrie J, and Philip J Guo. 2019. ”Software developers learning machine learning: Motivations, hurdles, and desires”. In *2019 IEEE symposium on visual languages and human-centric computing (VL/HCC),* 25–34. IEEE. https://doi.org/10.1109/VLHCC.2019.8818751.

Chandler, Paul, and John Sweller. 1996. ”Cognitive load while learning to use a computer program”. *Applied cognitive psychology* 10 (2): 151–170. https://doi.org/10.1002/(SICI)1099-0720(199604)10:2<151::AID-ACP380>3.0.CO;2-U.

Chung, Sungwon, Jongpil Cheon, Cristina Diordieva, and Jue Wang. 2015. ”The application of the segmenting principle: The effects of pause time and types in instructional animations”. *thannual,* 54. https://eric.ed.gov/?id=EJ1031133.

Clarke, Paul, Rory V O'Connor, and Brian Leavy. 2016. ”A complexity theory viewpoint on the software development process and situational context”. In *Proceedings of the International Conference on Software and Systems Process,* 86–90. https://doi.org/10.1145/2904354.2904369.

Fakhoury, Sarah, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. 2018. ”The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load”. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC),* 286–28610. https://doi.org/10.1145/3196321.3196347.

Galy, Edith, Magali Cariou, and Claudine Mélan. 2012. ”What is the relationship between mental workload factors and cognitive load types?” *International Journal of Psychophysiology* 83 (3): 269–275. ISSN: 0167-8760. https://doi.org/https://doi.org/10.1016/j.ijpsycho.2011.09.023.

Gerjets, Peter, Katharina Scheiter, and Richard Catrambone. 2004. ”Designing instructional examples to reduce intrinsic cognitive load: Molar versus modular presentation of solution procedures”. *Instructional Science* 32 (1): 33–58. https://www.jstor.org/stable/41953636.

Greer, Lindred L, Bart A de Jong, Maartje E Schouten, and Jennifer E Dannals. 2018. "Why and when hierarchy impacts team effectiveness: A meta-analytic integration." *Journal of Applied Psychology* 103 (6): 591. https://doi.org/https://doi.org/10.1037/apl0000291.

Guba, Egon G, Yvonna S Lincoln, et al. 1989. *Fourth generation evaluation.* Sage. ISBN: 9780803932357.

Helgesson, Daniel, Daniel Appelquist, and Per Runeson. 2021. "A Grounded Theory of Cognitive Load Drivers in Novice Agile Software Development Teams". *CoRR* abs/2107.04254. https://doi.org/10.48550/arXiv.2107.04254.

Helgesson, Daniel, Emelie Engström, Per Runeson, and Elizabeth Bjarnason. 2019. "Cognitive Load Drivers in Large Scale Software Development". In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE),* 91–94. https://doi.org/10.1109/CHASE.2019.00030.

Hoda, Rashina, Norsaremah Salleh, and John Grundy. 2018. "The Rise and Evolution of Agile Software Development". *IEEE Software* 35 (5): 58–63. https://doi.org/10.1109/MS.2018.290111318.

Hogg, Michael A, and Amber M Gaffney. 2018. "Group processes and intergroup relations". *Stevens' Handbook of Experimental Psychology and Cognitive Neuroscience* 4:1–34. https://doi.org/10.1002/9781119170174.epcn414.

Jansen, Harrie, et al. 2010. "The logic of qualitative survey research and its position in the field of social research methods". In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research,* volume 11. 2. https://doi.org/10.17169/fqs-11.2.1450.

Janssen, Jeroen, and Paul A Kirschner. 2020. "Applying collaborative cognitive load theory to computer-supported collaborative learning: towards a research agenda". *Educational Technology Research and Development,* 1–23. https://doi.org/10.1007/s11423-019-09729-5.

Johnson, David W, and Roger T Johnson. 2009. "An educational psychology success story: Social interdependence theory and cooperative learning". *Educational researcher* 38 (5): 365–379. https://doi.org/10.3102/0013189X09339057.

Kalkhoff, Will, and Christopher C. Barnum. 2000. "The effects of status-organizing and social identity processes on patterns of social influence". *Social Psychology Quarterly* 63:95–115. https://doi.org/10.2307/2695886.

Kirschner, Femke, Fred Paas, and Paul A Kirschner. 2009. "A cognitive load approach to collaborative learning: United brains for complex tasks". *Educational psychology review* 21 (1): 31–42. https://doi.org/10.1007/s10648-008-9095-2.

———. 2011. "Task complexity as a driver for collaborative learning efficiency: The collective working-memory effect". *Applied Cognitive Psychology* 25 (4): 615–624. https://doi.org/10.1002/acp.1730.

Kirschner, Paul A., John Sweller, and Richard E. Clark. 2006. "Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching". *Educational Psychologist* 41 (2): 75–86. https://doi.org/10.1207/s15326985ep4102.

Kirschner, Paul A., John Sweller, Femke Kirschner, and Jimmy Zambrano R. 2018. "From Cognitive Load Theory to Collaborative Cognitive Load Theory". *International Journal of Computer-Supported Collaborative Learning* 13:213–33. https://doi.org/10.1007/s11412-018-9277-y.

Klepsch, Melina, Florian Schmitz, and Tina Seufert. 2017. "Development and Validation of Two Instruments Measuring Intrinsic, Extraneous, and Germane Cognitive Load". *Frontiers in Psychology* 8:1997. ISSN: 1664-1078. https://doi.org/10.3389/fpsyg.2017.01997.

Lee, Joy Yeonjoo, Jeroen Donkers, Halszka Jarodzka, Géraldine Sellenraad, and Jeroen J.G. van Merriënboer. 2020. "Different effects of pausing on cognitive load in a medical simulation game". *Computers in Human Behavior* 110:106385. ISSN: 0747-5632. https://doi.org/https://doi.org/10.1016/j.chb.2020.106385.

Levine, John M, Hoon-Seok Choi, and Richard L Moreland. 2003. "Newcomer innovation in work teams". *Group creativity: Innovation through collaboration,* 202–224. https://doi.org/10.1093/acprof:oso/9780195147308.003.0010.

Lin, Chieh-Peng. 2010. ”Understanding Negative Impacts of Perceived Cognitive Load on Job Learning Effectiveness: A Social Capital Solution”. PMID: 21284366, *Human Factors* 52 (6): 627–642. https://doi.org/10.1177/0018720810386606.

Lincoln, Yvonna S, and Egon G Guba. 1985. *Naturalistic inquiry*. sage. ISBN: 9780803924314.

Massey, Aaron K, Richard L Rutledge, Annie I Antón, and Peter P Swire. 2014. ”Identifying and classifying ambiguity for regulatory requirements”. In *2014 IEEE 22nd international requirements engineering conference (RE),* 83–92. IEEE. https://doi.org/10.1109/RE.2014.6912250.

Matthews, Jamie. 2019. *Programming and Cognitive Load.* https://www.dabapps.com/blog/cognitive-load-programming/.

Mayer, Richard E, and Roxana Moreno. 2003. ”Nine ways to reduce cognitive load in multimedia learning”. *Educational psychologist* 38 (1): 43–52. https://doi.org/10.1207/S15326985EP3801_6.

Mayer, Richard E, and Roxana Ed Moreno. 2010. ”Techniques that reduce extraneous cognitive load and manage intrinsic cognitive load during multimedia learning.”, https://doi.org/10.1017/CBO9780511844744.009.

Mayer, Richard E, and Celeste Pilegard. 2005. ”Principles for managing essential processing in multimedia learning: Segmenting, pretraining, and modality principles”. *The Cambridge handbook of multimedia learning,* 169–182. https://doi.org/10.1017/CBO9781139547369.016.

Miller, George A. 1956. ”The magical number seven, plus or minus two: Some limits on our capacity for processing information.” *Psychological review* 63 (2): 81. https://doi.org/10.1037/h0043158.

Moreland, Richard L, and John M Levine. 1989. ”Newcomers and oldtimers in small groups.”

Moreno, Roxana Ed, and Babette Park. 2010. ”Cognitive load theory: Historical development and relation to other theories.”, https://doi.org/10.1017/CBO9780511844744.003.

Nowell, Lorelli S., Jill M. Norris, Deborah E. White, and Nancy J. Moules. 2017. "Thematic Analysis: Striving to Meet the Trustworthiness Criteria". *International Journal of Qualitative Methods* 16 (1): 1609406917733847. https://doi.org/10.1177/1609406917733847.

Orru, Giuliano, and Luca Longo. 2019. "The Evolution of Cognitive Load Theory and the Measurement of Its Intrinsic, Extraneous and Germane Loads: A Review". In *Human Mental Workload: Models and Applications,* edited by Luca Longo and M. Chiara Leva, 23–48. Cham: Springer International Publishing. ISBN: 978-3-030-14273-5.

Paas, Fred, Alexander Renkl, and John Sweller. 2003. "Cognitive load theory and instructional design: Recent developments". *Educational psychologist* 38 (1): 1–4. https://doi.org/10.1207/S15326985EP3801_1.

Paas, Fred GWC, and Jeroen JG Van Merriënboer. 1993. "The efficiency of instructional conditions: An approach to combine mental effort and performance measures". *Human factors* 35 (4): 737–743. https://doi.org/10.1177/001872089303500412.

Retnowati, Endah, Paul Ayres, and John Sweller. 2017. "Can collaborative learning improve the effectiveness of worked examples in learning mathematics?" *Journal of educational psychology* 109 (5): 666. https://doi.org/10.1037/edu0000167.

Roschelle, Jeremy, and Stephanie D. Teasley. 1995. "The Construction of Shared Knowledge in Collaborative Problem Solving". In *Computer Supported Collaborative Learning,* edited by Claire O'Malley, 69–97. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-85098-1. https://doi.org/10.1007/978-3-642-85098-1_5.

Schloegel, Uta, Sebastian Stegmann, Alexander Maedche, and Rolf Van Dick. 2018. "Age stereotypes in agile software development–an empirical study of performance expectations". *Information Technology & People,* https://doi.org/https://doi.org/10.1108/ITP-07-2015-0186.

Stettina, Christoph Johann, and Werner Heijstek. 2011. "Necessary and Neglected? An Empirical Study of Internal Documentation in Agile Software Development Teams". In *Proceedings of the 29th ACM International Conference on Design of Communication,* 159–166. SIGDOC '11. Pisa, Italy: Association for Computing Machinery. ISBN: 9781450309363. https://doi.org/10.1145/2038476.2038509.

Sweller, John. 1988. ”Cognitive load during problem solving: Effects on learning”. *Cognitive science* 12 (2): 257–285. https://doi.org/10.1016/0364-0213(88)90023-7.

———. 1994. ”Cognitive load theory, learning difficulty, and instructional design”. *Learning and instruction* 4 (4): 295–312. https://doi.org/10.1016/0959-4752(94)90003-5.

———. 2010. ”Element interactivity and intrinsic, extraneous, and germane cognitive load”. *Educational psychology review* 22 (2): 123–138. https://doi.org/10.1007/s10648-010-9128-5.

Sweller, John, Jeroen J.G. van Merrienboer, and Fred G. W. C Paas. 1998. ”Cognitive Architecture and Instructional Design”. *Educational Psychology Review,* https://doi.org/10.1023/A:1022193728205.

Tajfel, Henri. 1974. ”Social identity and intergroup behaviour”. *Social science information* 13 (2): 65–93. https://doi.org/10.1177/053901847401300204.

Tashtoush, Yahya, Zeinab Odat, Izzat M Alsmadi, and Maryan Yatim. 2013. ”Impact of programming features on code readability”, https://doi.org/10.14257/ijseia.2013.7.6.38.

Tavares de Souza, Alberto Luiz Oliveira, and Victor Hugo Santiago Costa Pinto. 2020. ”Toward a Definition of Cognitive-Driven Development”. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME),* 776–778. https://doi.org/10.1109/ICSME46990.2020.00087.

Teasley, Stephanie D, and Jeremy Roschelle. 1993. ”Constructing a joint problem space: The computer as a tool for sharing knowledge”. *Computers as cognitive tools,* 229–258.

Tobin, Gerard A, and Cecily M Begley. 2004. ”Methodological rigour within a qualitative framework”. *Journal of advanced nursing* 48 (4): 388–396. https://doi.org/10.1111/j.1365-2648.2004.03207.x.

Tuckman, Bruce W. 1965. ”Developmental sequence in small groups.” *Psychological bulletin* 63 (6): 384. https://doi.org/10.1037/h0022100.

Tuckman, Bruce W, and Mary Ann C Jensen. 1977. ”Stages of small-group development revisited”. *Group & organization studies* 2 (4): 419–427. https://doi.org/10.1177/105960117700200404.

Uludag, Ömer, Martin Kleehaus, Christoph Caprano, and Florian Matthes. 2018. ”Identifying and Structuring Challenges in Large-Scale Agile Development Based on a Structured Literature Review”. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC),* 191–197. https://doi.org/10.1109/EDOC.2018.00032.

Van Merrienboer, Jeroen JG, and John Sweller. 2005. ”Cognitive load theory and complex learning: Recent developments and future directions”. *Educational psychology review* 17 (2): 147–177. https://doi.org/10.1007/s10648-005-3951-0.

Van Merriënboer, Jeroen JG, Liesbeth Kester, and Fred Paas. 2006. ”Teaching complex rather than simple tasks: Balancing intrinsic and germane load to enhance transfer of learning”. *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition* 20 (3): 343–352. https://doi.org/10.1002/acp.1250.

Wang, Xiaoran, Lori Pollock, and K Vijay-Shanker. 2011. ”Automatic segmentation of method code into meaningful blocks to improve readability”. In *2011 18th Working Conference on Reverse Engineering,* 35–44. IEEE. https://doi.org/10.1109/WCRE.2011.15.

Appendices

A Haastateltavan tausta (only in Finnish)

1 Voitko kertoa omasta taustastasi ja kokemuksestasi kehittäjänä? Mitä teet nykyäään? Lyhyt esittely riittää

2 Jos ajatellaan niitä aikoja, kun aloitit nykyisessä tiimissäsi:

B Ryhmä- ja tiimityöskentely (only in Finnish)

1 Kerrotko lyhyesti, millaisessa tiimissä aloitit? Mikä oli oma roolisi?

2 Kertoisitko omin sanoin tiimin työskentelytavoista?

3 Mitkä asiat sai työskentelyn sujumaan tai mitkä asiat helpottivat työskentelyä?

4 Koitko, että jokin toimi erityisen hyvin tiimityöskentelyssä? Kertoisitko esimerkin?

5 Entä mitkä asiat vaikeutti työskentelyä? Kertoisitko omin sanoin?

6 Koitko joitain haasteita tiimityöskentelyssä?

C Työskentelyvälineet (only in Finnish)

1 Mitä työvälineitä tiimissäsi käytetään? Mitkä ovat kaikilla käytössä ja mitkä saat valita itse? (esim. IDE, versionhallinta-alusta yms, työnseuranta-alusta?)

2 Kun tulit tiimiin, oliko sinulla aikasempaa kokemusta käytetyistä työvälineistä (esim. Git, Jira, VersionOne, yms.)

3 Jos mietitään alkuvaihetta, oliko mikään näistä haastava ottaa käyttöön? Mikä teki siitä haastavaa?

4 Oliko jokin hyvin helppoa? Mikä siitä teki helpon?

D Informaation saatavuus ja selvyys (only in Finnish)

1 Jos sinulle tuli tai tulee jotain kysyttävää, miten lähdit ensimmäisenä etsimään tietoa?

2 Oliko tarvittava tieto helposti saatavilla? Miksi, tai miksi ei?

3 Oliko etsimäsi tieto helppo sanoittaa kysymykseksi?

4 Millä tavoin etsit tietoa?

5 Millaisista asioista nousi eniten kysyttävää?

E META (only in Finnish)

1 Muita esille tulevia asioita?

2 Miten haluaisit, että perehdytys tiimiin tehtäisiin sinun kohdallasi? Mistä asioista hyvä perehdytys koostuu?

3 Muita tiimiin tulemisen liittyviä asioita, joita haluat sanoa:

4 Millaiseksi itse kuvailisit kokemasi kognitiivisen kuorman?

5 Lievittääkö tiimin tuki kognitiivista kuormaa omasta mielestäsi?