Aalto-yliopisto
Aalto-universitetet
Aalto University

Master's Programme in Computer, Communication and Information Sciences, Software and Service Engineering

Case study: Findings from applying an actionable developer experience framework

Antti Harju

Master's thesis
2024

Aalto-yliopisto
Aalto-universitetet
Aalto University

Author Antti Harju

Title of thesis Case study: Findings from applying an actionable developer experience framework

Programme Computer, Communication and Information Sciences

Major Software and Service Engineering

Thesis supervisor Professor Fabian Fagerholm

Thesis advisors Marko Teittinen, M.Sc. Telecommunication; Hannu Heinilä, M.Sc. Telecommunication

Collaborative partner Nokia Solutions and Networks Oy

Date 15.9.2024 **Number of pages** 51 **Language** English

Abstract

Developer experience is a branch of research related to user experience. As a concept, it can include inspecting how developers perceive, feel, or see various aspects of their work. This thesis lays its foundations on a recently published actionable developer experience framework.

This thesis aims to understand how developers in a real software development team perceive developer experience factors from the actionable framework. During the thesis process the team's tooling was also further developed.

Data for the analysis was collected through qualitative interviews with the six software developers in the case team, further supported by the author's participant observations as the seventh developer. The aforementioned tooling development is described to gather understanding on what improving developer experience can mean for practitioners.

The thesis presents various findings about the framework. The main contributions are figures that illustrate how software developers perceive the developer experience factors in the actionable framework, based on which developers can be interviewed with increased precision. The framework was found to be mostly descriptive, although it was presented with actionable advice. A foundation for a more actionable framework is presented.

The results require further validation using quantitative methods. Other directions for further research, for example cause-effect relationships related to developer experience, are also presented.

Keywords case study, developer experience, framework, actionability, software engineering, qualitative interview

Aalto-yliopisto
Aalto-universitetet
Aalto University

Tekijä Antti Harju

Työn nimi Tapaustutkimus: Löydöksiä toiminnallisen kehittäjäkokemuksen viitekehyksen käytöstä

Koulutusohjelma Computer, Communication and Information Sciences

Pääaine Software and Service Engineering

Vastuuopettaja/valvoja Professori Fabian Fagerholm

Työn ohjaajat Marko Teittinen, M.Sc. Telekommunikaatio; Hannu Heinilä, M.Sc. Telekommunikaatio

Yhteistyötaho Nokia Solutions and Networks Oy

Päivämäärä 15.9.2024 **Sivumäärä** 51 **Kieli** Englanti

Tiivistelmä

Kehittäjäkokemus tutkimussuuntana liittyy käyttäjäkokemuksen tutkimukseen. Sen tutkimuksessa voidaan tarkastella esimerkiksi, miten kehittäjät havainnoivat, tuntevat, tai näkevät erilaisia heidän työhönsä liittyviä asioita. Tämä tutkielma pohjaa hiljattain julkaistuun toiminnallisen kehittäjäkokemuksen viitekehykseen.

Tämä tutkielma pyrkii ymmärtämään miten kehittäjät oikeassa ohjelmistokehitystiimissä havainnoivat toiminnallisen viitekehyksen kehittäjäkokemuksen osatekijöitä. Tutkielman aikana tiimin työkaluja kehitettiin kehittäjäkokemuksen parantamiseksi.

Data analyysia varten kerättiin haastattelemalla laadullisesti kuusi samassa tiimissä olevaa ohjelmistokehittäjää. Lisäksi kirjoittaja on tehnyt havaintoja osallistumalla tiimin toimintaan seitsemäntenä ohjelmistokehittäjänä. Aiemmin mainittu työkalujen kehitys on myös kuvattu ymmärryksen kartuttamiseksi siitä, mitä kehittäjäkokemuksen parantaminen voi tarkoittaa ammatinharjoittajan näkökulmasta.

Tutkielma esittää useita löydöksiä viitekehyksestä. Pääkontribuutiot ovat kuvioita, joilla havainnollistetaan miten ohjelmistokehittäjät havainnoivat kehittäjäkokemuksen osatekijöitä viitekehyksestä, joiden avulla kehittäjiä voidaan haastatella entistä tarkemmin. Viitekehys havaittiin pitkälti kuvailevaksi, vaikka sen ohessa tarjotaan toimintaan johtavia ohjeita. Tutkielma esittää perustan toiminnallisemmalle viitekehykselle.

Tulokset vaativat pidemmälle vietyä validointia määrällisillä metodeilla. Tutkielma tarjoaa myös muita suuntia jatkotutkimukselle, esimerkiksi kehittäjäkokemukseen liittyvät syy-seuraussuhteet.

Avainsanat tapaustutkimus, kehittäjäkokemus, viitekehys, toiminnallisuus, ohjelmistotuotanto, laadullinen haastattelu

Preface and acknowledgements

First and foremost, I want to thank Nokia for offering the thesis worker position. Nokia was represented by the thesis advisors Marko Teittinen and Hannu Heinilä, who I also want to thank for allowing me to work on a self-proposed topic and for their input from the formulation of the thesis all the way to the finishing touches.

I also want to thank the supervising Professor Fabian Fagerholm for his input regarding developer experience research. Overall, his advice throughout my master's studies has been invaluable.

Thanks also to my friends, especially Samuli Mannonen for his support and Hyosun Kim for her love during the thesis process.

Finally, I want to thank my family for raising me with the work ethic I have and for their understanding and support.

Espoo, July 16, 2024
Antti Harju

Table of contents

1 Introduction ... 7
1.1 Motivation ... 7
1.2 Research questions ... 8
2 Background ... 10
2.1 Developer experience ... 10
2.2 An actionable developer experience framework ... 12
3 Research method ... 15
3.1 Interview guideline ... 15
3.2 Participant selection and interview execution ... 15
4 Perceptions of developer experience factors ... 16
4.1 Classifying the factors ... 16
4.2 Discussing the factors ... 20
4.2.1 High importance ... 20
4.2.2 Medium importance ... 23
4.2.3 Low importance ... 26
4.2.4 Divisive ... 27
4.3 Factor clarity ... 31
4.4 Connections between factors ... 33
4.5 Possibly missing aspects ... 35
5 What improving developer experience can mean in practice ... 36
5.1 Tooling improvements ... 36
5.2 Process improvements ... 39
5.3 Further work ... 40
6 Discussion ... 41
6.1 Limitations ... 44
6.2 Further research ... 45
7 Conclusions ... 46
References ... 47
A Interview guideline ... 49
B Developer experience factors and categories ... 51

1 Introduction

The goal of this thesis is to investigate how practitioners consider developer experience, as suggested by Nylund (2020, p. 60). The thesis also describes tooling improvements to ground the thesis in a real context.

Developer experience—from now on mostly referred to as DX for brevity—in academic literature draws parallels from user experience or UX research (Fagerholm & Münch, 2012, p. 74). In DX the set of users is just limited to developers building software. This leads to developer tooling—for example software related to continuous integration—being a prime target for usability improvements in pursuit of improving DX, instead of the end user product. The overarching idea is still to produce a better product for the end user, through a “virtuous cycle” (Meier, 2015) as illustrated in Figure 1 below.

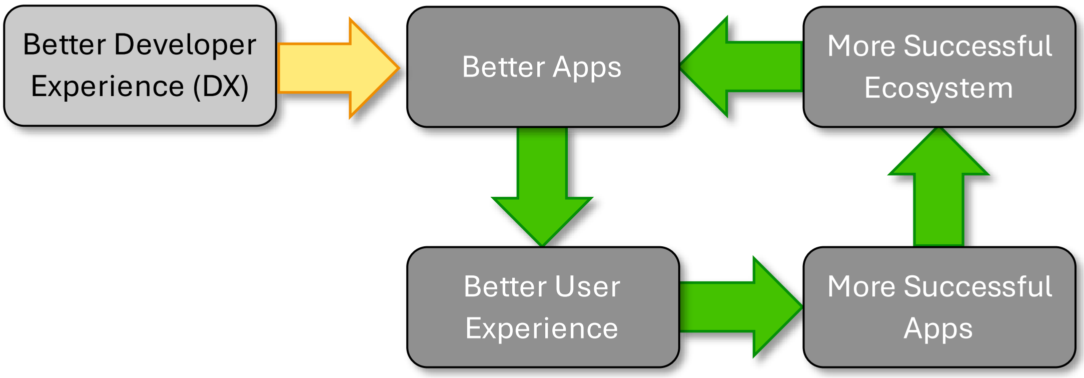


Figure 1. A virtuous cycle of DX (adapted from Meier, 2015).

1.1 Motivation

It has been argued that for academic research to “overpower the folklore, fads, or industry “best practices” that often guide operations” (Siponen and Baskerville, 2018, p. 247) it must be able to demonstrate intervention effectiveness. In other words, ideally research would lead practice instead of describing it.

A major enabling factor for attempting to apply academic research in an industrial setting was the thesis worker position offered by Nokia, which led to the author joining a software development team as a developer. While the position mostly consisted of carrying out practical tasks to help the team reach their goal of bringing a new product to market, the tasks were often made to fit an “improving developer experience” perspective in collaboration with the lead developer. Additionally, soon after having agreed upon the topic with the thesis advisors, a week was dedicated to establishing the basic structure of the thesis. The qualitative interviews were also carried out during work hours.

In the upcoming background section on DX, research papers demonstrating intervention effectiveness were not identified, but one presenting "an actionable framework" (Greiler et al., 2023) was identified as a key reference. While other interesting papers were also found, the framework seemed the most promising for the goal of this thesis, as it appeared to have been formulated with practitioners in mind, offering further structure for the case.

This thesis describes how the software developers in the case team perceived the developer experience factors from the actionable framework by Greiler et al. (2023). Additionally, practical tasks related to improving DX—implemented without employing the use of the framework—are also described along with suggestions for further improvements. The descriptions hopefully help future research reach for the much higher-level goals such as measuring DX and using the measurements as a basis to demonstrate intervention effectiveness. For example in cancer research, one measure is the two-year survival rate after applying treatment (Siponen & Baskerville, 2018, p. 256). The actionable framework seems to fit well with these aspirations; "we anticipate that the DX Framework may be used as a foundation for systematic approaches for assessing and improving developer experience" (Greiler et al., 2023, pp. 1423). Demonstrating intervention effectiveness within DX research could mean things such as providing estimates for how much applying a technique could increase employee retention over a two-year period.

1.2 Research questions

Before presenting the research questions, it is necessary to briefly introduce the aforementioned DX framework (Greiler et al., 2023, p. 1416). The framework will also be introduced in section 2.2 in more detail.

In short, the DX framework can be used to describe aspects affecting DX. As a short example, a junior developer's (contextual characteristic: seniority) cognitive performance may be reduced due to an *unreasonable deadline* (developer experience factors are *italicised*), which may have been set in the first place due to a lack of awareness of the junior developer's skills. The junior developer may then end up working overtime, after which they could employ an individual strategy of speaking up about issues with their task, or a team strategy of creating transparency to their daily work to make their struggles more visible in order to receive help.

Again, the framework will be described in more detail later. What is relevant for now is that the framework can be used to describe and speculate about scenarios, as demonstrated above.

According to the framework the DX factors together with contextual characteristics form DX. This, and the fact that the original paper had seemingly placed a lot of effort on formulating the factors, made it sensible to pick the factors as a starting point for the research questions.

Therefore, the first research question is **RQ1** "how do the software developers in the case team perceive the developer experience factors from the framework?". In other words, do the developers in the case team understand the factors consistently, and what connections do they draw between the factors. These questions will be explored and answered in chapters 4 and 6. Given the relative freshness of the original paper (Greiler et al., 2023), a closer inspection of the framework seems essential before employing its use further.

The second research question is **RQ2** "where do practitioners direct their attention when they want to improve developer experience?". During this thesis, practical tasks with the general purpose of "improving developer experience" were carried out. The tasks were practitioner-led, i.e. no research method was applied, but with the later descriptions of the tasks the thesis aims to provide tangible examples of what improving developer experience at an organisation can mean.

2 Background

2.1 Developer experience

A starting point for the research was a paper by Fagerholm and Münch (2012) where developer experience is split into three parts: cognition, affect, and conation. The paper explains that this parallels the definition from psychology where cognition corresponds to "attention, memory, producing and understanding language, problem-solving, decision-making", affect to "feeling, emotion", and conation to "impulse, desire, volition, striving" (Fagerholm & Münch, 2012, p. 75). On the same page in figure 1, the paper also presents a more approachable way to express these dimensions with the following three questions:

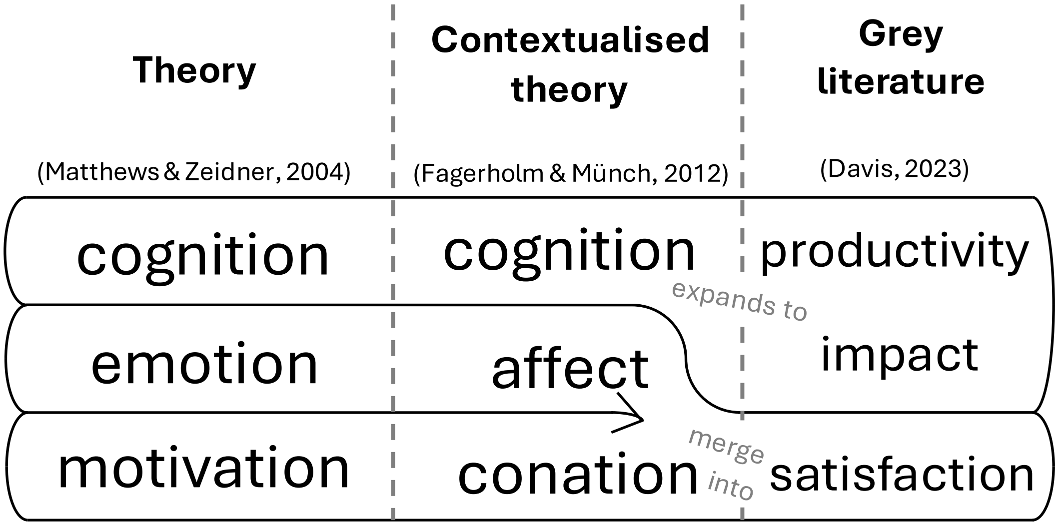
1. Cognition: How do developers perceive the development infrastructure?
2. Affect: How do developers feel about their work?
3. Conation: How do developers see the value of their contribution?

An alternative definition for developer experience from grey literature is offered by Davis (2023) where developer experience is summarised with three parts: productivity, impact, and satisfaction. These terms are explained as follows:

1. Productivity: how quickly or simply a change can be made to a codebase
2. Impact: how frictionless it is to move from idea to production
3. Satisfaction: how the environment, workflows, and tools affect developer happiness

This definition also happens to be the one favoured by the latter thesis advisor, also holding the role of the lead developer in the case team. It is what they looked up independently when they wanted to get to know the topic of developer experience.

An initial research question for this thesis was how much the affective and conative dimensions of the definition by Fagerholm and Münch (2012) would arise in the interviews with the software developers in the team, when focusing on the cognitive parts of their work, that is, the development infrastructure. However, this type of question seems to not have yielded interesting results earlier, for example Greiler et al. (2023) observed that they "anticipated that we might be able to align the [DX] factors that would emerge from our study with the three dimensions of the trilogy of mind but we did not find this". Greiler et al. (2023) further continue that this finding was in line with a book section by Matthews and Zeidner (2004) where it is claimed that "it is necessary to treat cognition, emotion, and motivation as inextricably related", which further supported abandoning the initial research question.



Unfortunately, it does not seem possible to locate this direct quote in the referenced book.

Nevertheless, the book still offers a possible explanation for the cognitive focus in the grey literature observed for example by Nylund (2020, p. 30), "we will also argue that cognition is the most fundamental of the trilogy for understanding individual differences" (Matthews & Zeidner, 2004, p. 145). In other words, if there is one part of the trilogy of mind that offers the most understanding on individual differences—which can be helpful in improving developer experience—it would make sense for practitioners to focus on it, even if the affective and conative dimensions are also essential for forming the overall experience. This fits well with the observation by Nylund (2020, p. 30) where "scientific articles were dwelling more equally on cognitive, affective, and conative, the three aspects of DX while grey literature was clearly more focusing on the cognitive parts of DX".

Another way to illustrate the different emphases is by mapping the three dimensions of the definitions to each other. The productivity and impact dimensions of the definition by Davis (2023) seem to map best to the cognitive part of the definition by Fagerholm and Münch (2012) as it is characterized by the question "how do developers perceive the development infrastructure?". Similarly, the satisfaction part of the definition (Davis, 2023), seems to map best to the dimensions affect and conation (Fagerholm & Münch, 2012), per their respective characterizing questions of "how do developers feel about their work?" and "how do developers see the value of their contribution?". Figure 2 illustrates the different taxonomies and emphases placed on DX.

Figure 2. Developer experience from trilogy of mind to grey literature through contextualized theory.

2.2 An actionable developer experience framework

At this point it is essential to properly discuss the DX framework (Greiler et al., 2023, p. 1416), available in Figure 3. The authors describe the framework as suitable for "framing theoretical propositions about how certain interventions may be used to improve developer experience" (Greiler et al., 2023, p. 1423). In other words, the framework seems to provide a shared language to be used with the actionable advice near the end of the paper. For utilising the framework Greiler et al. (2023, pp. 1423–1424) propose an "Ask-Plan-Act" Process. It is summarised with the following steps:

- "*Ask: Make problems visible.* The first step to assessing developer experience is to learn about the day-to-day experiences of developers. As factor importance is highly dependent on the responsibilities, activities, and goals of each developer, it is crucial to ask each individual developer about their experience. Our factors and barriers can be used as prompts for collecting feedback through structured feedback mechanisms (e.g. surveys) or unstructured methods (e.g., retrospectives, one-on-one meetings).
- "*Plan: Improvement needs to happen on an individual, team and organizational level.* Once developer feedback has been collected, it can be analysed to determine the areas that need to be improved. To ensure that individuals and teams are empowered to make improvements, we recommend assigning explicit owners for each improvement area. Improvement efforts need to be planned for and be allocated appropriate time and resources."
- "*Act: Continuous, small improvements are key.* To work on improvements, developers and teams may benefit from working incrementally with fast feedback loops. Strategies from the DX Framework, such as building bridges, convincing others and creating transparency, may help overcome barriers to improvement. Developers can use individual strategies to make improvements themselves without relying solely on their teams or leaders to drive change. The factors can also be used to derive measures for assessing developer experience and monitoring the success of improvement efforts. This makes both the problems and improvement efforts more visible. Once planned improvements have been implemented, this three-step process can be repeated to continue progress on a continuous basis."

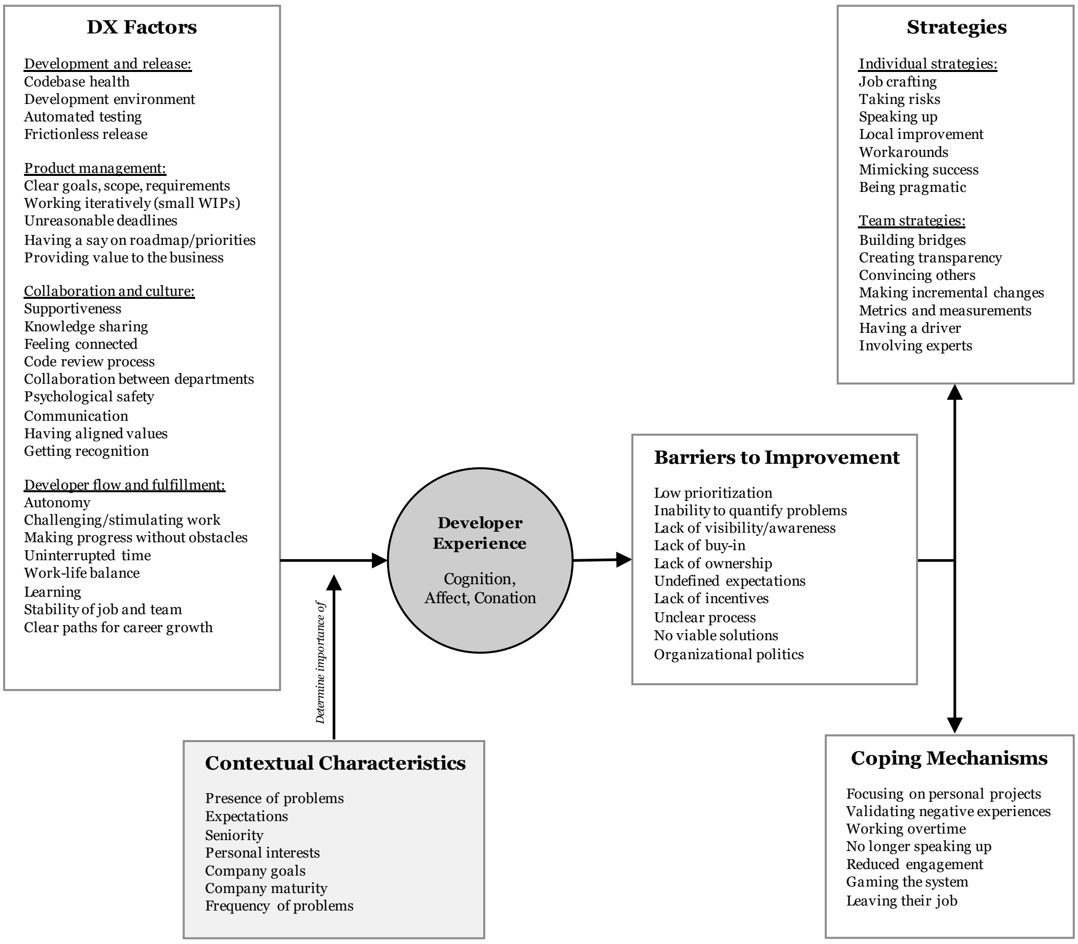


Figure 3. Developer experience framework (Greiler et al., 2023, p. 1416).

In the introduction it was demonstrated how the framework in Figure 3 can be used to describe scenarios. Here is another short scenario: a developer is *learning* (DX factor) a lot due to the presence of problems (contextual characteristic), which they see as valuable (conation).

Before fully utilising the DX framework (Greiler et al., 2023), it seems essential to check whether its language is understood consistently by the subject developers. Martin Fowler for example describes in his blog post related to Domain-Driven Design (Evans, 2004) how “different groups of people will use subtly different vocabularies in different parts of a large organisation” (Fowler, 2014). As an example, he mentions the words *customer* and *product* across sales and support contexts. In the support context both words are associated with a ticket, while in a sales context with a territory. This is illustrated in Figure 4.

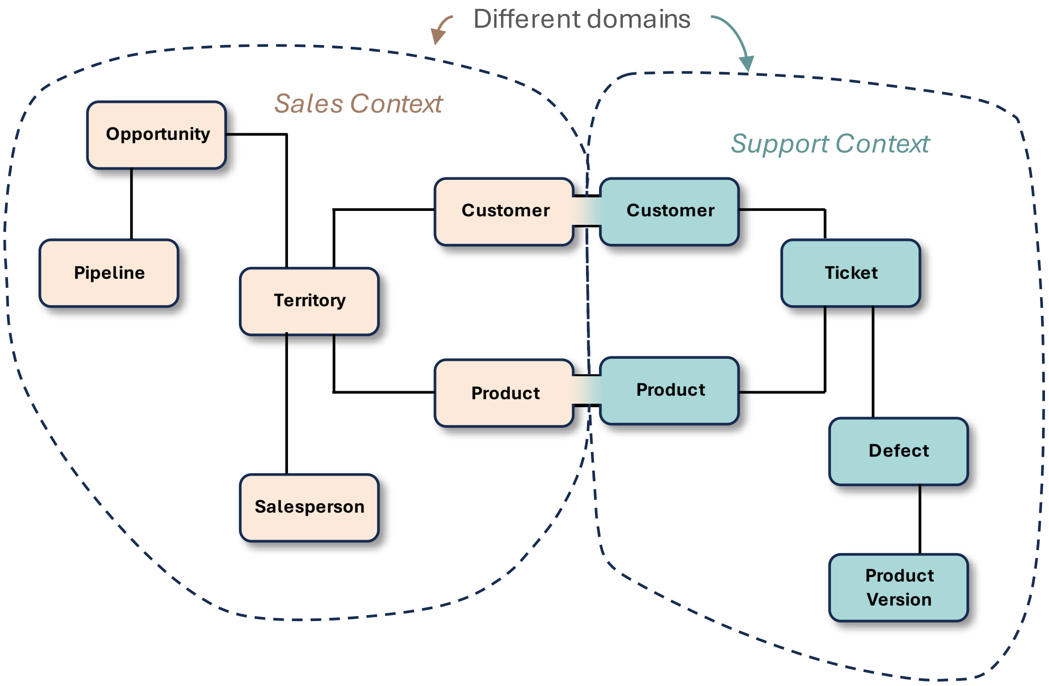


Figure 4. Examples of bounded contexts (adapted from Fowler, 2014).

As the framework may have been formulated with developers from a wide range of domains, this thesis will cover whether the language in the framework is understood consistently by the six developers in the case team to assess construct validity. This will be explored in section 4.2.

3 Research method

Semi-structured interviews were opted for as a qualitative research method, to parallel the original paper (Greiler et al., 2023). The interview guideline used in the original paper (Greiler & Noda, 2022) was used as a template, with the difference that instead of trying to evaluate factors collected from a literature review, the factors were those presented in the final paper.

3.1 Interview guideline

In the first part of the adapted interview guideline (available in Appendix A) the participants were introduced to DX as defined by Davis (2023) and given the opportunity to voice their unguided impressions about the definition. This also served as a way to give the participants time for context switching.

In part 2, instead of asking "why" a participant rated a specific factor the way they did, they were asked "what led them" to their answer. This hopefully invited more elaborate answers. The participants were rather talkative, producing in total 78 pages or 26 643 words of transcribed text from more than 281 minutes of interview recordings.

Retained from the original interview guideline, the participants were given the opportunity at the end of the interview to point out if the interview missed any important aspects, also yielding interesting answers.

3.2 Participant selection and interview execution

The interview subjects were picked as a convenience sample, i.e., the six software developers in the case team. The participants gave informed consent for interview participation. The interviews were optional to participate in, although the developers received explicit encouragement from the team's manager. Thankfully all software developers in the team participated. The participants also had the option to not answer some questions to encourage more candid answers. The interviews were one-on-one (vs. a group setting) for the same reason.

All interviews were conducted in person in roughly 45-minute sessions using pen and paper for rating the factors in Appendix B. The participants had the option of replying in either Finnish or English as the author is a native Finnish speaker. Any non-English answers were translated.

While the interview results are anonymised, the promise of anonymity presents a potential problem, as the participants likely know each other professionally relatively well. As each developer who participated in the interview may end up reading this thesis, they could be able to piece together who answered and how. Therefore, the answers per participant per DX factor are disconnected from each other to increase the difficulty of piecing together the answers to avoid identification.

4 Perceptions of developer experience factors

This chapter focuses on answering **RQ1** "how do the software developers in the case team perceive the developer experience factors from the framework?". First the factors are classified based on how the case software developers perceived their importance as a team. After that each factor will be described based on the individual answers, followed by sections on factor clarity, connections between the factors, and possibly missing aspects.

Demographic information about the participants is presented in Table 1 below. The table aims to reflect the contextual characteristic of seniority from the DX framework in Figure 3, as the figure claims that contextual characteristics "determine the importance of" the DX factors. Tenure is based on the time the participant has been part of the current team—all participants belong to the same team—and total experience is based on the time the participant has worked in any software development team. Both are self-reported. Participant order is random and does not match the order in which the following sections discuss the interviews.

Participant	Tenure	Total experience
P1	8 months	15 months
P2	28 months	30 years
P3	11 months	25 years
P4	12 months	2 years
P5	18 months	3 years
P6	1 month	7 months

Table 1. Overview of participant seniority.

4.1 Classifying the factors

This section classifies the DX factors based on the importance placed on them by the case team to provide a more meaningful structure for the next section. The classifications here do not consider possible differences in how each factor was understood, which will be discussed in the later sections 4.2, 4.3, and 4.4. Rather, the figures in this section are just for illustrative purposes to help one understand the kind of data available.

The participants were interviewed individually, and during the interviews they were asked to rate the perceived importance of each DX factor as either low, medium, or high. The factors are listed in Appendix B. The classifications presented here are specific to the case team and separate from categories such as "development and release" and "product management" from the original research paper (Greiler et al., 2023). The categories are visible in Figure 3. All ratings are represented in Figure 6.

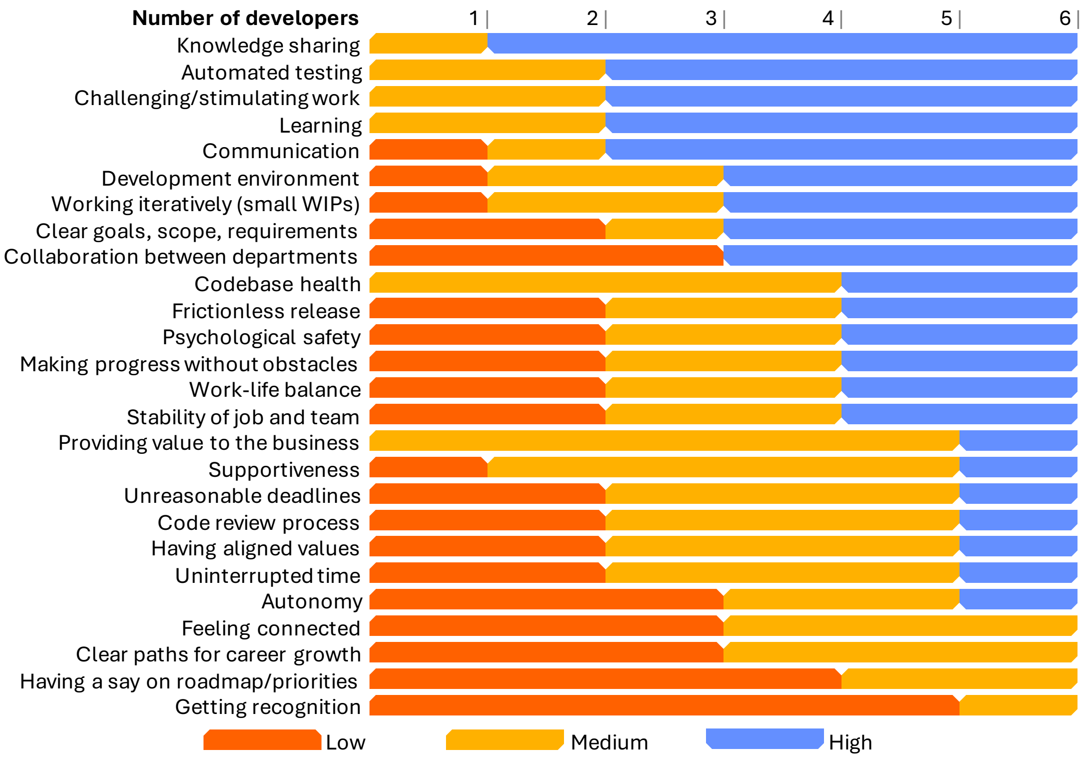


Figure 6. Developer experience factors as rated by the case team.

No DX factor received a completely unanimous importance rating, although *knowledge sharing* and *getting recognition* came close, being the most important and least important DX factors respectively. The factor *collaboration between departments* stands out as the most divisive factor, half the team giving it a high rating and the other half a low one. It will be separated into its own classification.

While the data in Figure 6 is sorted, it is easier to understand if the factors that received three types of importance ratings are temporarily omitted. The resulting figure, Figure 7, contains the factors the team is most in agreement about, but also the one the team is most divided about. Individual factors will be elaborated upon in the subsections of section 4.2, but Figure 7 still provides the most relevant ones if one wants to understand what is or is not important to the case team.

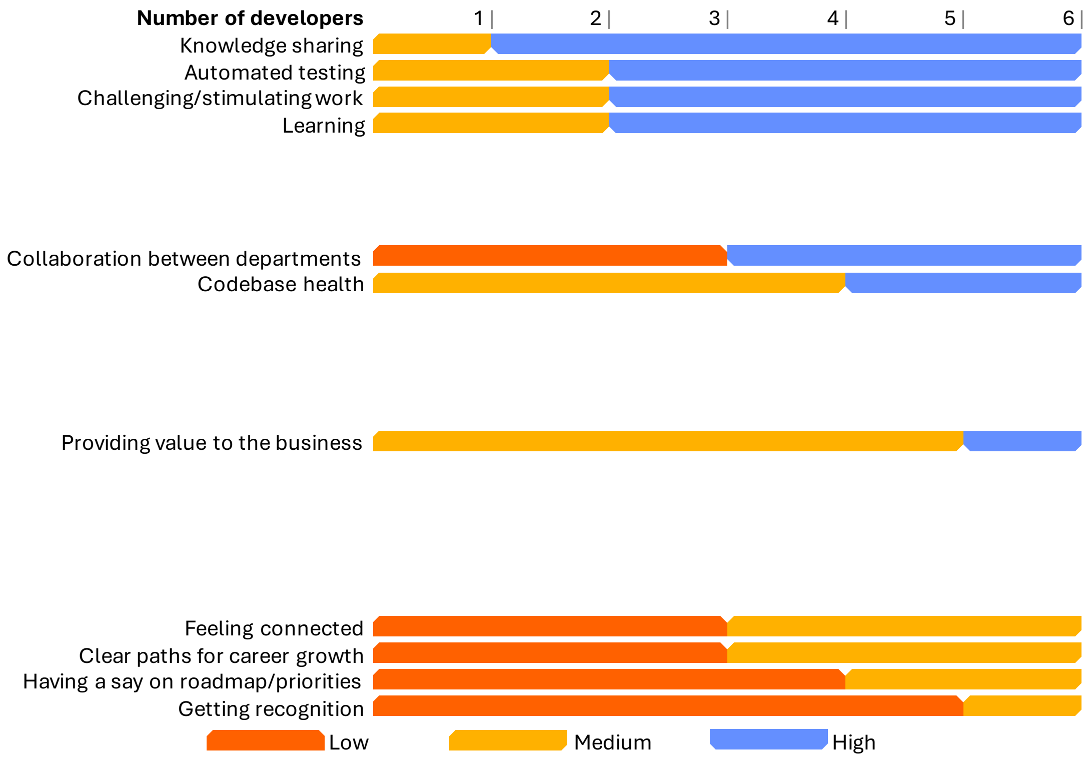


Figure 7. Factors with only two types of ratings.

Before moving on to analysing the individual factors, it is also helpful to view the factors omitted from Figure 7. These are in Figure 8, where it becomes apparent that several factors received only one rating that pushed it from Figure 7 to Figure 8. These can be considered outliers and included in the dominating importance rating's classification. A handful of factors also received an equal number of each importance; therefore, they will be included in the divisive classification.

Classifying the factors like this makes strong assumptions about whether the participants understood the classifications consistently, but the goal with this is to just provide more structure for section 4.2. The final classifications are presented in Figure 9.

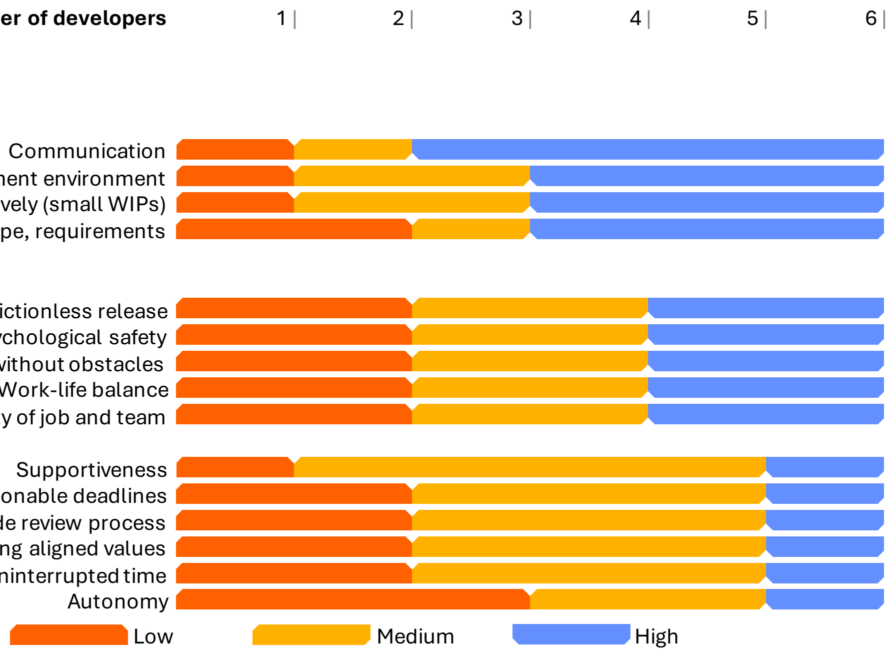


Figure 8. Factors with three types of ratings.

Figure 9. Factor ratings classified for the case team.

4.2 Discussing the factors

Having introduced the contextual characteristic of seniority and the factor classifications in the preceding sections, each DX factor per classification can be described in detail based on the interviews.

The first paragraph per factor aims to provide a summary of the apparent dimensions in the answers, while the following ones describe each dimension in more depth. The summary is by the author, while the following paragraphs mostly stick to what the interviewees described; the author may still bring in related but unmentioned terminology or literature references. Nevertheless, it should be clear from the text whether the text stems from the author or an interviewee. Within this section, direct quotes from participants are marked using double quotation marks “”, without a reference ().

It should be emphasized that this section describes how the participants view the importance of each factor as individuals, also using their previous experiences. Therefore, the answers described here are not necessarily an assessment of their current environment.

4.2.1 High importance

Knowledge sharing M H H H H H was seen as the most important because it helps avoid wasting time and it makes the team less reliant on individual developers.

A participant described how sometimes developers may waste time just because they did not know about a new feature in the system under development. Another participant described how some tasks may be completely trivial to one developer, while requiring a lot of time from another. They gave an example of how it may be difficult to investigate bug reports from the testing team, if the report has unfamiliar logs. Through knowledge sharing, the developer initially investigating the ticket can contact the one who most recently worked on a related feature, especially if they had warned during knowledge sharing it may have issues. Some participants simply answered that knowledge sharing enables them to be more productive.

What is commonly referred to as the bus factor was also brought up. Bus factor is about managing the risk of one developer suddenly leaving the team, in the namesake example by being hit by a bus. The participants listed a person switching jobs, getting sick, or being busy as concrete examples. One participant further elaborated on how if only one person has expertise in a certain area, situations may arise where the whole team stalls on progress when the person is unavailable.

Automated testing M M H H H H was seen as a prerequisite to enter a competitive market and overall as very important, with some caveats. The answers seemed to mostly focus on the aspects of automated tests saving time and giving confidence when writing new code.

One caveat for the importance was how for example 100% code coverage is not necessary. Rather, it was deemed more important to test the requirements of a feature in terms of “here’s what we set out to do and here’s the [test] code, and the code does what we tried to do”. Another caveat was how automated tests do not necessarily imply tests that cover the existing functionality well.

An aspect brought up by several participants was how automated tests save time. One participant did napkin math on how for example if one lands 300 merge requests in two years and if manual testing each takes two hours, that is already 600 hours. Dividing that by 8 hours a day is 75 full working days and if there is 250 working days a year, it leads to one spending 15% of their time just on testing. Another participant pondered upon automated vs. non-automated testing and concluded that while manual tests are in theory possible, lack of automated tests may in practice lead to developers not testing their code as much.

Another aspect was being able to gain confidence in changes, by receiving a quick feedback loop from automated tests. They were seen as helpful when working in the less familiar areas of the codebase. This was emphasised by one of the participants who was relatively new to the team.

Challenging/stimulating work M M H H H H was seen as the engaging aspect of the work. It was described as leading to increased productivity, but also as the absence of routine. The participants seemed to dislike routine.

A few participants described *challenging/stimulating work* as the fun part of the job. One participant reasoned how if the work is not challenging it is basically “boring and not fun”, and they want to do fun things. They acknowledged that of course sometimes one needs to do not-fun things, but it is something they saw as important in general. Another participant described this aspect as a way of keeping their mind active.

The same participant also speculated that probably almost everyone in the software industry views this aspect as important, as otherwise they might have ended up with a different type of job, using assembly line worker as an example. Further support for this came from a few other participants, one describing how they dislike routine as it does not allow them to *learn*. Another viewed this factor as a “basic principle of a [software development] job” through which one may enter a so-called “flow state”, becoming more productive.

Learning M M H H H H was seen as highly related to *challenging/stimulating work*, although the relationships between the different factors will be discussed in a later section. Viewing this factor as important was described as a prerequisite for the role of a software developer and while it was also described as an engaging part of the job, all answers made the relationship from *learning* to job security or career.

Again, *learning* was described as “the fun part”, with the question that “if you’re not *learning*, you’re not really doing anything interesting”.

Coincidentally, this same thought is echoed by Raph Koster in his book "A Theory of Fun for Game Design" where he presents that "fun is just another word for *learning*" (Koster, 2004, p. 46). But at the same time, the lack of *learning* was seen as a threat to one's job security by a participant, because routine tasks can be automated.

This same line of thinking was supported by other participants where they saw *learning* as a means for career growth, one describing how lack of *learning* could lead to "a stalemate or straight line" for their career. Another participant—with a chuckle—saw *learning* as a means of getting more money as their career progresses. A third participant stated that as they are at an early point in their career, it is a given that *learning* is important.

Communication L M H H H H is the first factor in this classification that includes an outlier importance rating, as will all the subsequent ones. There were some differences in understanding it, although factor clarity will only be briefly touched upon here and a more thorough dissection is available separately in section 4.3. Nevertheless, most of the team perceived the factor as important.

A few participants described how lack of *communication* may lead to people not knowing what is going on and ending up doing duplicate work, leading to for example backtracking. One participant described how if a change was not what was requested, it gets frustrating for everyone quite quickly.

Although the previous paragraph seems to better suit the factor *knowledge sharing* as one participant described how *communication* is not necessarily fact-based like *knowledge sharing* but is more about what is happening in general. Another participant explained how good *communication* outwards from the team "starts from the grassroot-level of having good *communication* in between our team members first". Overall good *communication* was seen as a precondition to *knowledge sharing*, *supportiveness*, and *psychological safety*.

On the other hand, a participant described how an infinite amount of effort can be put into *communication* and quoted an emeritus professor on how "*communication* usually fails, except by accident" ("Osmo Antero Wiio", 2024). This quote—despite its popularity—is not without its critics, for example Kortesuo (2013) has suggested alternatives, viewing failure by default as cynical. Regardless, the participant did not view *communication* as a crucial aspect for the team's overall success.

Development environment L M M H H H seemed to be understood consistently, but the different importance ratings boiled down to individual preferences. Nevertheless, most of the team perceived the factor as important.

One participant summarised their low importance rating as that while a good Integrated Development Environment (IDE) may be beneficial for their productivity, tooling does not matter to them as an individual as long as it is possible to get the job done.

Another participant listed requirements such as easy to use, intuitive and stable as requirements for their IDE as it is the place where they spend most of their time working. But the participant seemed to have a similar attitude as the previous one as soon as the definition expanded outside the IDE: “Build machines, [hardware] test equipment, etc.” just need to “do their jobs”.

A third participant expanded the importance beyond the IDE as all of it speeds up their work, spending considerable portion of their answer describing what may be required for a proper development environment setup. They listed things such as installing a relevant SDK, Docker, kubectl, minikube, and configuring proxy access to individual programs.

Working iteratively (small WIPs) L M M H H H seemed to be like *development environment* in the sense that while important, the differences seemed to again boil down to individual preferences rather than differences in understanding the factor.

One participant stated that task size (for example in terms of lines of code) does not matter to them. On the other hand, another participant felt that the team often works on what they would describe as a black box, and for development related to it, “it’s nice to have little proof of concepts and work in progresses” before trying to do something big and realizing that it does not work. Another participant emphasised how they can give better time estimates for smaller tasks vs. big ones.

4.2.2 Medium importance

Providing value to the business M M M M M H was the most medium importance factor for this team. Nevertheless, it received one high importance rating where a participant questioned what they would be doing if they were not *providing value to the business*. Although they did further continue to think out loud how some tasks may provide value in a less direct way. When queried for examples, they mentioned writing tests. The participant described how automated tests do not necessarily have direct value, “obviously there are benefits in the long term”.

Supportiveness L M M M M H is the first factor in this classification that includes outliers, that is, one low and one high importance rating. This factor included some slight differences in understanding: one answer focused on being able to receive help and one more on how possible problems are handled.

The low importance was reasoned with that while it is nice to get help if the participant is stuck with a task, they will figure it out themselves should no help be available. On the other hand, a high rating was based on that a participant appreciates it when no individual is blamed for mistakes and the team tries to improve together to make the product work. They summarised the answer with that they enjoy being in a "*supportive* environment where we are a team, not individuals".

Codebase health M M M M H H was one of the factors that, despite its medium importance rating, received elaboration in the final thoughts section from one participant and received some high importance ratings which further help the description here.

A participant described how *codebase health* is obviously important, especially in the long-term, but on the other hand if "everyone acknowledges that the situation is what it is, and then you have to make some non-optimal decisions to get things done", it is not as important in the short term.

Another participant based their high importance rating on the fact that they saw it as a prerequisite to enter a competitive market. Another one discussed how a codebase with low health would result in "many other problems" and how a healthy codebase would allow some mistakes.

Unreasonable deadlines L L M M M H leaned towards a low importance rating, but also had plenty of mediums and one high. It had some differences in understanding the factor. Ambitious deadlines could have been a better word to describe one participant's answer, while a second participant mostly focused on the word deadline itself, and a third one on the word unreasonable. This factor also raised the most questions about how to answer using the options of low, medium, and high, due to it being phrased negatively where the others are either positive or at least neutral. The participants were instructed that if they feel this factor could affect them a lot, they should rate it as high.

This one also received further elaboration from one medium importance rating: "*unreasonable deadlines* are important to have, but not in an unreasonable manner". The participant continued to explain how when deadlines are set and one sometimes hits them "you're probably quite happy with yourself, having achieved something that you didn't know that you could get done". An alternative phrasing of "ambitious deadlines" was suggested to the participant, and they agreed it better described what they meant.

A low importance rating was reasoned through that "deadlines, whoever sets them, are always more or less negotiable. The holy trinity: content, schedule, or quality, and one of them always gives in".

A high rating was reasoned through how in general no one probably enjoys *unreasonable deadlines*. The participant also emphasized that they had not encountered them in their current team.

Code review process L L M M M H also leaned on low importance rating but had received plenty of mediums and one high. This factor was understood mostly consistently, two answers focusing on the process itself and one more on the code reviews happening in the first place.

One participant based their low importance rating on how they felt the *code review process*—in general and the one employed in the current team—did not have much of an effect on the quality of the code. They also wondered if the team was spending too much time on them compared to the achieved benefit. Nevertheless, they acknowledged it as a good way to onboard new developers to the team.

This last view gained support from one of the more junior interviewees who perceived the importance as high and described how "good reviews, actually getting feedback" is valuable to them compared to "someone only having a superficial look at the code and pressing approve".

Another participant reasoned how it is something that must be done one way or another, and the end result is roughly the same. Therefore, for them the process itself "doesn't really influence" them as an individual.

Having aligned values L L M M M H received mostly medium importance ratings, with a few lows and one high importance rating. Lows tended to stem from a very high-level understanding of the word value, while highs from a more practical one, for example values related to coding practices.

One participant based their low importance rating on how "the values in their head are relatively rigid and the ones in Nokia's operations match their expectations well". The participant said how the rating could also have been high, but then they had paid attention to the word "aligned" and as they perceive the company as engineering-led, alignment is not something they really need to think about, because the values are so ingrained to the company. Another participant based their low importance rating on how values, for example political, do not necessarily matter as long as they are able to work with their colleagues.

The high importance rating was reasoned through how aligned values "can significantly speed up your *review process*" because "if you have aligned values, you probably generate similar solutions". This would then lead to less time discussing with the colleague what each line of code should be, as they would need to be "ready to support them [the lines of code] together in the future".

Uninterrupted time L L M M M H also received mostly medium ratings, with a few lows and one high. It was understood mostly consistently, but the differences boiled down to individual preferences.

One participant who rated it as high, felt that "at least two hours [of *uninterrupted time*] is a bare minimum". Their definition of uninterrupted did not mean that no one could ask them anything and it was more about not having to sit in meetings. The participant stated that half-hour breaks between meetings—including lunch—cause the most inefficiency.

This matched well with another participant who had rated the factor importance as low, because they felt interruptions "in the sort of psychological level it breaks up the thinking process. Or even just on a physical level you get some different input into your system that changes your thinking and maybe you have some different angle on the problem, and you realize you have been doing something wrong. So, I see often it is a positive thing". A third participant based their low importance that their more senior role in the team prevents them from requiring *uninterrupted time*.

4.2.3 Low importance

Getting recognition L L L L L M was the lowest importance factor for this team. It seemed to boil down to individual preferences.

One participant described "I don't care about *recognition*. I know what I did and if I'm happy with what I did, then that's fine. Nobody needs to know it was me who did it". Another participant described how when they "do something, it's for the team and for the product", mentioning how they do not require praise.

A third participant elaborated on their thoughts more. "If I think that something went badly, it doesn't matter if somebody else tells me it was good. The same applies the other way around: if I don't get *recognition* for something I did well, it doesn't matter, but if I get *recognition* for something and I am not so sure if it's deserved, that's not helpful either".

Having a say on roadmap/priorities L L L L M M seemed to be rated as low due to participants not viewing it as a part of their job description as software developers.

A participant described how they did not feel like it was a part of their role, but further elaborated that they would not be pleased if they could not voice their opinion when something seems wrong. They speculated that with more experience this factor could rise in importance, should it be part of their job description.

Another described how they "probably wouldn't even want to *have a say on roadmap or priorities*. Depends of course if it's my features". When queried about whether there could be some scenarios where this factor would be important, they stated the following: if "we have two tasks, one is a fix to some bug and one is a feature: a bigger change that completely removes the component that has the bug. Then if for some reason the prioritisation is to fix the bug and then delete the code, that seems silly and would be annoying". Although the participant immediately followed with that as they like to solve problems, they would probably enjoy even this situation.

A third participant stated that "as long as they're allowed to state if something's not doable, it's enough". On the other hand, they described that if a product is in such a state that everything that has been done has been necessary, this factor would lose its relevance.

Clear paths for career growth L L L M M M gained little bit longer answers, but nevertheless continues the trend of short ones. The low importance seemed to boil down to individual preferences rather than differences in understanding the factor.

One participant described that "I like to focus on what I'm doing now, so it's not that important that I see 10 years into the future and okay I will do this work two years and then I will advance to this". "I'll see how I feel about things in the future and then I'll decide what I do and that's fine". Another reasoned that they were not looking for more career growth. The participant was already well into their career.

Feeling connected L L L M M M also received relatively short replies. One participant based their low importance rating on "let's say if my co-workers didn't want anything to do with me, just only the minimum possible interaction then it's fine". Another reasoned similarly, "I wouldn't feel bad if I'm unconnected, just a personal preference".

Autonomy L L L M M H importance ratings were based on individual preferences, for example by preferring to collect best practices from others.

One participant emphasised the importance of adopting best practices from other teams around them, leading them to rate it as low importance. But not all ratings were low though, as another participant rated this factor as high because they "enjoy working independently".

A third participant gave a longer description of *autonomy*. While they rated its importance as low, they emphasized that they did not see it "as something really good or something bad". In their response they focused on how *autonomy* may lead to situations where only the *autonomous* person has the knowledge on several components and is the only one able to support them. They further continued how asking others for help is important, as "you can make your work faster and do your work with more quality and results, because most probably your colleagues will share with you best practices". Their conclusion was that "I don't see any reasons to be really *autonomous* if you're working in a team".

4.2.4 Divisive

Collaboration between departments L L L H H H was the most divisive factor. Some participants seemed to dismiss it based on either that the team does not have much of a need to participate in it or on individual preferences. The ones who rated it as high, reasoned that it is an important factor to maintain productivity for the "complicated system" the team is developing.

One participant described how our team all things considered has relatively little of it. They described how while we have collaboration with one offshore team, that collaboration is fairly standardised and works well, therefore they placed low importance on this factor.

Another participant reflected on the factor through their individual preferences, so this collaboration does not seem to be something that matters to them. Although, their viewpoint seemed to focus more on general discussion rather than task-based interactions. If another department is working on something that affects their daily tasks, then they would have some interest.

But some participants rated this factor as high. Third participant described how "it's important that the collaboration is good, so that the productivity is high". The participant placed special focus on possible disconnects between departments as "it gets really annoying quite quickly" leading to "not getting things done".

A fourth participant continued to rate the factor as high. They described how "the end goal is kind of to integrate our microservice or endpoint to another service or vice-versa, then it would be great or nice that we could have those communication lines open to other teams to enable fast iteration and just in general for knowledge sharing". They were especially concerned about the cases where "the other components or microservices are kind of stateful and depend on the others like requests or responses, so the communication of what should happen and both parties should be very informed and the only way how to inform the other is to fix the communication between those parties".

There was even a fifth participant who rated it as high. They based their answer on how "we're working on the complicated system with many components and many teams developing these components". "Whenever you want to implement a completely new feature and it's possible that feature will affect all or most of them [...] you need to have quite good *collaboration between different departments* so you can share your work, what every team is supposed to implement for that feature, and if you don't have good collaboration, then it will take a lot of time to do so".

Frictionless release L L M M H H the participants who rated it as low saw it as not part of their job description, while the ones who rated it as high focused on how it might be a way to gain loyalty from customers or the lack of it could cause business stakeholders to lose their patience.

A participant who rated this as low reasoned their answer through how even "if you have a very complicated and long process for getting releases done, it doesn't really at the end of the day affect what you do next. You still just make the change and then the process is what it is. Maybe it takes longer, but I don't feel like that's my problem". Another participant followed similar reasoning where "as I'm not working with releases, this factor is not so relevant for me".

A third participant rated it as high and took the stance that unless one has a *frictionless release* process, it means that one is not able to get releases done and business stakeholders will lose their patience.

Another participant took *frictionless release* process to mean that the team can deliver any patch in quite short time to testing team or production. They further elaborate how "critical bugs are always coming from the customer and then we need to be able to deliver the hotfixes in a short time. Such ability is quite important to support customer to have a bit loyal customer. This is better than when you need to say like 'you need to wait two weeks until our fixes ship to your system'. To the testing teams, to the verification, etc., such possibility is important".

Psychological safety L L M M H H seemed to receive high ratings when the participants were considering team dynamics, and low ratings based on individual preferences.

One participant initially rated it as low, but as they were asked to elaborate on their answer, they first upped it to medium, and finally to high. Their reasoning was that while they could not see themselves ever being in a situation where it would matter to them, i.e. they would always ask questions if they felt like asking, but then they began to consider how for example if a colleague would struggle with the factor to the point that they would not like to come to work, it could start affecting them as well. Another participant who also rated it as high, explained how "it's important that the working space has no psychological bullying or something like that".

On the low rating side, one participant based their rating on how "in this industry one needs to be able to tolerate uncertainty and uncomfortable feelings". They had already worked for some time, so they saw this as something that came to them naturally.

Making progress without obstacles L L M M H H seemed to depend on what the participants considered an obstacle. The different classifications did not seem to be unified in their dimensions. In other words, each answer seemed to have a unique perspective.

One participant rated this factor with high importance as they saw the factor as an indicator of a project's good state, meaning the codebase is written well, tested well, and it is prepared to be extended with new features. Another participant who also rated the factor as high, first pondered about the term itself as it seemed vague. They reasoned that if an obstacle is solvable, it is not really an obstacle. Therefore, they went on to explain that obstacles could be things that for example should not happen in a job culture. As an example they described how one can produce a solution for a problem, but someone else might not understand them. This may lead to an obstacle where the solver needs to wait in a stalemate until the other one understands it on their own, even if one has tried to explain the problem.

A third participant who rated the factor as low, reasoned that given our current project integrates both hardware and software "there's nothing but obstacles". They further elaborated that making progress without obstacles is something that would more likely occur in pure software projects, such as game development [the software the team is working on focuses on hardware]. A fourth participant who also rated it as low, explained how "if there's obstacles, then it's just *challenging and stimulating work*!" with a chuckle. But on a more serious note, they pondered how even if they get stuck, there is always another task they can continue with.

Work-life balance L L M M H H seemed to be based on individual preferences. For some it happens automatically, while others try to avoid conditions where it could be at risk.

One participant who rated it as low, added a precondition that they need to “like what I’m doing for work”. They explained how working 12 hours a day for several weeks is not good, but further based their rating on how working a lot is not something which is allowed in their job. Another who rated it as low felt work-life balance was achieved automatically for them.

A third participant rated it as high, as they had experienced the lack of work-life balance. “It just leads to the downfall of your own well-being, health, and eventually the work itself [...] in the long run one achieves the best result through work-life balance”. Another who also rated it as high explained it is a precondition when they are applying for a job.

Stability of job and team L L M M H H seemed to have high ratings placing attention on people leaving, and low ratings on lack of control over the factor or individual preferences.

One participant explained their high rating through a general perspective, stating “it’s probably the most pleasant for anyone if the team around you is not cycling all the time”. “It also sucks if one is very afraid whether they’ll lose their job or not, so this doesn’t really relate to developing specifically”. Another participant dissected the factor, explaining how “it doesn’t matter so much if new people are coming in, but if there are people leaving at a high velocity, it affects everything the team does, and usually not in a good way [...] we lose competence, and the work becomes more difficult”.

Another participant stated that both job and team are something they have no control over, so they do not place importance on them. A fourth one said they do not care about stability and thus have nothing else to state.

Clear goals, scope, requirements L L M H H H one participant rated low based on an individual preference where “they’re quite capable of living without clear goals and scope, mainly because they can figure them out on their own to the extent required”.

Another participant also rated the factor as low, as they saw this factor as strongly related to the waterfall method and not the agile methods the team follows. They stated that the waterfall method would not be suitable for this product.

However, a third participant rated this factor as high. They wanted to be specific that they were not referring to requirements, “but more like the developer daily tasks that you get”. Their main concern was how the product must comply with a specification and how someone who is not familiar with said specification might produce requirements for tasks that are not compatible with the specification. Later the participant added the existing microservice architecture as another justification for their high rating.

A fourth participant also emphasised the importance of clarity with the factor, as otherwise they are not able to provide time estimations on the tasks. They added how unclear requirements may increase the time spent on reviews, where additional requirements may be introduced due to the lack of clarity.

4.3 Factor clarity

While the previous section already lists some dimensions each factor may have, this section goes over the instances when the participants asked what an individual factor means. Overall, the participants' ratings did not seem contradictory as the differences could be explained by the different emphases placed on the factors. This section attempts to distil those emphases from the answers so that in the future more specific questions may be asked about the factors from the DX framework (Greiler et al., 2023).

At the start of each interview, all factors were read out loud to the participant and they were prompted to ask for clarifications before rating the perceived factor importance, see the interview guideline in Appendix A. Table 2 below contains the factors that were asked about when rating the factors in Appendix B.

Factor	Clarification requests
Unreasonable deadlines	3
Codebase health	2
Frictionless release	1
Making progress without obstacles	1
Psychological safety	1
Stability of job and team	1
Supportiveness	1

Table 2. Factor clarification requests in the interviews.

Unreasonable deadlines stood out as the factor raising the most questions. The former thesis advisor also immediately pointed it out as *prejudiced* (in Finnish: *asenteellinen*). In the original paper it is framed as a part of product management, "causing poor trade-offs in terms of timeline, roadmap, and priorities [...] that many participants described as stressful and negatively impacting developer experience" (Greiler et al., 2023, p. 1417). If one wants to put a positive spin on it, it could be called *ambitious deadlines*, but this would still result in a leading question. Leading questions mean prompting "the interviewee through the use of words other than those used by the interviewee [which] can result in misleading answers" (DiCicco-Bloom & Crabtree, 2006, p. 40). Both ambitious and unreasonable are adjectives that may prompt the interviewees for certain types of answers, so to gain a better understanding on what are the interview participants' views on deadlines, it may be helpful to simply prompt for *deadlines*. Other factors in the framework seem relatively neutral—although positive adjectives such as clear and frictionless are used. Nevertheless, these seemed clear to the case team and omitting the adjectives could change the purpose of the factors significantly.

The rest of the factors in Table 2 seemed to pass with relatively little explanation. The original research paper (Greiler et al., 2023) offers descriptions for them, but these were not always helpful when the participants were asking what does it mean to rate a factor as a low importance one. In general, the interview participants were asked to rate the factors through how much they saw the factor as being able to affect them. In other words, rating for example unreasonable deadlines as low importance does not mean that they do not exist, but just that they are not something the participant necessarily has to think about in their work.

In *psychological safety* for example one participant initially rated the factor as low when they were reflecting it on themselves, but gradually raised it to high as they began to reflect it on the team—they could see the factor affecting themselves if for example a team member experienced a lack of *psychological safety*. Taking it further, an organisation offering a psychologically safe work environment may attract applicants for open positions.

When considering *having aligned values*, a participant rated it as low when they were considering how well the organisation's values match theirs, but another rated it as high when they were considering how well their values match their team's. They saw *having aligned values with the team* as a way to speed up the *code review process*.

Usually, a participant reflected a factor on only one level. It is possible that some of the divisiveness would disappear if the participants were prompted using specific levels, i.e., taking an existing factor name and appending the desired level. Exact formulation will depend on how the interview questions are phrased, but as an example with the interview guideline used in this thesis (Appendix A), a question template could be phrased as *please rate the perceived importance of* [DX factor] [*from your perspective as an individual / from the team's perspective / from the organisation's perspective] as low, medium, or high*. It may be necessary to introduce all levels, so the participants are able to focus on a specific one.

The level could be decided to fit the research goals. It could for example be locked at *organisation* if the research wants to focus on employer attractiveness, at *team* if team dynamics are a focal point for the research, and at *individual* if the research wants to focus strictly on the developers as individuals. Locking to a single level could also be counterproductive as all levels affect developer experience. For example, the team level probably has a relatively big effect on DX compared to the organisation level, so even when individual developers are a focal point for the research, locking to the *individual* level may not be helpful.

In a semi-structured or unstructured interview format the levels could be used to prompt for more thorough answers, for example asking a participant to reflect *having aligned values* on the team level, when they rated it a low when reflecting it on the organisation level.

4.4 Connections between factors

Another interesting aspect in the interviews was how the participants would make connections between the DX factors, unprompted. This thesis only counts explicit mentions of two or more factors as connections. In other words, when a factor was related to something adjacent to another one, it was not counted as a connection. For example, *learning* is not connected to *clear paths for career growth* based on a participant making the connection from learning to just career growth (lacking the *clear paths* portion); see interview excerpt 1 below.

> Interviewer: "*Then the next high is **learning**, what led you to high?*"
>
> Participant: "*Well, I guess every person's [laughs] goal in life is to learn something new every day, and I would love to be in an environment where I could learn and improve and build a kind of **career growth** that goes upwards, and not like on a stalemate or straight line.*" ✗

Rather, the connections are only illustrated when at least one participant made the connection explicitly, as interview excerpt 2 below demonstrates.

> Interviewer: "***Making progress without obstacles**, low. What led you to low?*"
>
> Participant: "*Obstacles are no bother, don't have to make progress. If there's obstacles, then it's just **challenging and stimulating work**! [laughs] If there's obstacles then they must be solved.*" ✓

However, slight variations such as share the knowledge for *knowledge sharing* were counted. See an example in interview excerpt 3 below.

> Interviewer: "*You've rated **autonomy** as low, what led you to low?*"
>
> Participant: "*I guess I don't see autonomy as something really good or something bad, but my understanding then for an autonomous guy, it's possible that at some point he will less **share the knowledge** [...].*" ✓

Figure 10 illustrates the connections. Descriptions corresponding to the alphabetically labelled connections can be found in Table 3. Please note, the figure is not exhaustive, meaning that more connections could probably be made. An arrow always points to the factor that came up during the discussion of the source factor, see the interview excerpts above for examples.

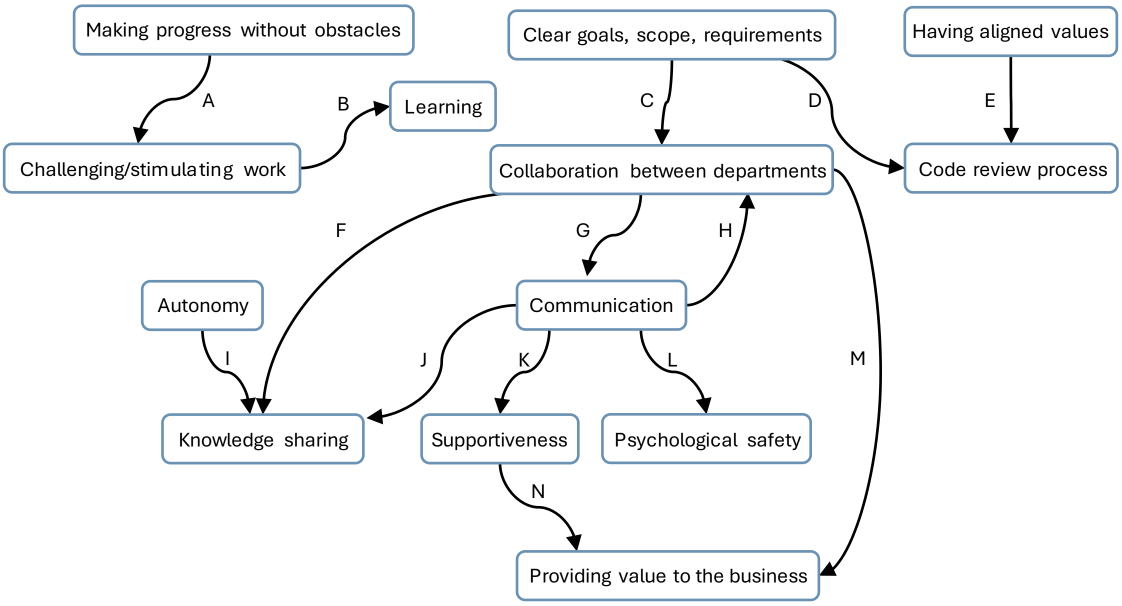


Figure 10. Explicit connections between the factors by the participants.

Arrow	How participant(s) made the connection
A	Solving obstacles is challenging/stimulating work
B	Challenging tasks allow one to learn more during implementation
C	Without clear requirements and working collaboration between the departments it is difficult to know what implications a task has.
D	Code review process slows down without clear requirements
E	With aligned values co-workers probably generate similar solutions, speeding up the code review process
F	Collaboration between departments is enabled through knowledge sharing
G	Collaboration between departments happens through communication
H	Good communication is a building block of collaboration between de-partments
I	Autonomous co-worker may share less knowledge with others
J	Knowledge sharing is a type of communication; Rating communication as high importance is a prerequisite for rating knowledge sharing with the same level.
K	Good communication is a building block of supportiveness
L	Good communication is a building block of psychological safety
M	It is important that collaboration between departments is good to be able to provide value to the business.
N	Without supportiveness one may sometimes spend a lot of time on a task. It is not the best way to provide value to the business.

Table 3. Explanations for the arrows in Figure 10.

4.5 Possibly missing aspects

At the end of part 2 of the interview guideline in Appendix A, the participants were asked "can you think of additional factors that are missing from this list (available in Appendix B) or that we haven't talked about that are important or significantly influence your developer experience?"

One participant used this as an opportunity to elaborate on their medium ratings for some factors and concluded "I think the list is quite comprehensive". Another participant simply answered, "maybe not" and used the opportunity to elaborate further on a factor they rated as high. A third one replied "no, nothing came to my mind, I guess. Good list [laughs]".

A fourth participant concluded the same with "this is very good, comprehensive list, what has been gathered here". They came up with the thought that the list does not prompt the participant for things such as "what kind of day would you like to work, are you an evening person or morning person, would you like to 90% remotely and 10% in the office". But the participant began to think how these would fit with working in a team, if everyone has different preferences.

A fifth participant suggested codebase size in addition to *codebase health*, as it can have an effect on how long it takes for a developer to become familiar with a codebase. They also mentioned domain knowledge, reflecting on how many new things such as specs or technologies there are to learn to work on a codebase. But the participant concluded that these may be relatively tough to affect as they pretty much depend on what an employee already knows or what kind of materials are available for learning the domain.

A sixth participant mentioned the maintenance perspective, meaning fixing bugs and lifecycle management. They saw for example different types of defects, customer issues, and new features affecting developer experience. The participant concluded that these are visible in the everyday work of a software developer working on a product. These could fit under the contextual characteristics in the DX framework presented in Figure 3, which was not presented to the participants. Characteristics *presence of problems* and *frequency of problems* seem to match well what the participant described, even if the lifecycle state of the software being worked on is not explicitly mentioned.

5 What improving developer experience can mean in practice

This chapter focuses on answering **RQ2** “where do practitioners direct their attention when they want to improve developer experience?” by describing the relevant practical tasks that were carried out during the author’s participant observation period in the case company. The tasks were often made to fit an “improving developer experience” perspective, with guidance from the latter thesis advisor who is also the lead developer in the case team. The goal in this chapter is to present a collection of practitioner-led DX improvements.

5.1 Tooling improvements

Dependency caching. In an effort to speed up the Jenkins pipeline, dependency caching was added to Dockerfiles. This was done through cache mounts (*Mounts*, n.d.). Without cache mounts, every time the Dockerfile is built, all external dependencies are downloaded which can take minutes per run. Exact figures for the time saved per run are difficult to obtain as the load on the build machine cannot be controlled, but for example running docker build on the laptop, dependency caching saved 65 seconds for the main Go module build and 154 seconds for the secondary, which is a significant portion of the whole pipeline run, which takes roughly 5 minutes. Similar time saves seemed to appear in the Jenkins pipeline runs.

But it should be kept in mind that there are no free lunches: “any two algorithms are equivalent when their performance is averaged across all possible problems” (Wolpert & Macready, 2005). In this case this means that any subsequent modifications to the Dockerfiles need to be made carefully. For example, when changing the access method to private external dependencies one has to remember to invalidate the cache, as otherwise the test runs may appear to work even if the new credentials do not work, thanks to the caching mechanism.

Making it possible to run unit tests locally. In the secondary service, a unit test had a hardcoded dependency to a configuration file location. This was copied to the correct location inside the Dockerfile, but not in the local development environment, unless a developer had set up a symbolic link in their filesystem. This hardcoded path was removed, and the tests used default values instead.

While most of the developers used Windows Subsystem for Linux (WSL), developing on Windows directly was also a supported use case. At some point the unit tests had begun failing on Windows, and they were fixed. The issues stemmed from not properly closing created files, causing the test cleanup phase on Windows to fail as it could not remove open files and from some of the code using platform-specific path separators for URLs, leading to the incorrect usage of backslashes \ when a forward-slash / should have been used.

Removing calls to external services in tests. Sometimes running the test suite would start failing continuously, meaning that all subsequent runs would fail too. Investigating the cause for this, some http GET related function tests were making calls to real websites, and should the website refuse to respond, the test would fail. Thankfully, Go's standard library already included a solution for this type of testing via httptest, which enables setting up an equivalent to a real HTTP server for testing purposes. Now the tests would succeed even if no internet connection was available.

Speeding up tests. The existing tests made extensive use of sleep calls. They are a straightforward way to write test code that succeeds consistently when different parts of the test are running concurrently in different threads. Some of these could be removed in favour using message channels and wait groups without polluting the production code with test-related parameters, each removal netting a second or so. While these may seem tiny, they were still significant improvements as when one runs the test suite locally, the duration was around 37 seconds.

Sometimes sleeps have to be used for testing whether timeouts work as expected. The default values were usually in seconds. In some cases, simply more seconds were used, but with relatively little effort the wait durations could be made configurable, and milliseconds be used instead. For example, the testable function times out in a millisecond, and the test code would wait for two.

Increasing test coverage. On several occasions, effort was put into simply increasing test coverage for existing test cases. Sometimes this included writing unit tests and at other times integration tests. Writing new integration tests also entailed writing message sequence charts above them, so that when another developer would have to have a look at it, they would be able to understand the test case at a glance.

Increased test coverage should be especially helpful for continuous integration. It helps developers be more confident in changes, as they would see whether they cause any existing code to break.

Preconfiguring the editor—a good idea? An attempt was made at including the build flag in the .vscode/settings.json file at the Go module root directories. For example, integration to the golangci-lint linter and a formatter were defined here. While these worked for some of the developers in the team, it did not work for all as in one case the new settings caused a developer's editor to freeze. Due to this, a premade decision to include .vscode/settings.json in version control was reversed, allowing the developers to customise the configuration file to fit their needs. It cannot be emphasised enough in this thesis how "developer experience is highly personal" (Greiler et al., 2023, p. 1423). Instructions were added to the Go module root readme on how to configure a git pre-commit hook to run the linter on commit.

Various logging improvements. The team is using Uber's zap logging package, which allows logs to have levels. Common levels include debug, info, warn, and error. With leveled logs, the desired level of information in the logs can be adjusted; for example, in a lab environment debug logs may be available while they are disabled in production. Info-level logs may provide general information about what the system is doing. Warnings are similar to info-level logs, but are used to log possibly problematic behaviour, such as cache misses. Errors are for things that should not happen and also produce a stack trace when they are called.

Certain stack trace appeared often in the logs, polluting them. The error logging was defined at the end of a switch-case statement, which was inside a function determining the type of message. The issue was solved by adding the missing cases. Log quality was also another area where work was carried out. For example, in general success cases would not be logged, but to log function call flows in logs, enter/exit logs were added. These were meant to log the function name, but with renames they would occasionally get out of date, and these would then have to be later fixed. Other times these enter/exit logs were removed from common getter functions as they would pollute the logs.

A common use case for inspecting logs is figuring out what the software has communicated with the TR-069 protocol in a half-duplex manner. The protocol sent SOAP messages that were labelled with "Send to" and "Received from" in the logs, with the IP address of the equipment in question. This made it somewhat cumbersome to read the logs, as the IP address was not seen as meaningful in itself. The situation was improved via adding a unique label in front of each sent and received message, with the serial number of the equipment in question. Serial numbers were seen as more meaningful and with the unique label it should be easier to go through the protocol communications in the log file quickly.

Various linter fixes. As one of the first tasks issues reported by a variety of linters available under golangci-lint were fixed. In general, linters are useful as they help developers produce higher quality code on their own, leaving less work for the code reviewers. Issues for linters such as whitespace, misspell, predeclared, wastedassign, staticcheck, unconvert, nosprintfhostport, dupword, godot, gofmt, gofumpt, protogetter, tagalign, testifylint, inamedparam, gocritic, asciicheck, bidichk, contextcheck, gocheckcompilerdirectives, and gochecksumtype were fixed. Details on what each linter does can be found online.

Whitespace consistency requires further context: the software product the team is working on focuses on the communication between it and hardware devices. This communication can be characterised as half-duplex, i.e., a hardware device sends a message, the server responds, and so on until the session is complete.

Integration tests focus on testing this communication against equipment mocks. To help developers understand what an individual test case does at a glance, message sequence charts (MSCs) are used. They are illustrated using ASCII arrows in code comments to have them viewable next to the test code.

However, the MSCs had inconsistent whitespace, meaning, the alignment of different arrows was produced with a mix of spaces and tabs. While spaces are of consistent width, tabs are not, i.e., a developer may configure them to be as wide as they like. Common values include 2, 4, and 8 spaces wide. This meant, that while the arrows in an MSC are aligned on one developer's computer, they would be misaligned on another's.

To fix the issue, a .editorconfig (*EditorConfig*, n.d.) file was introduced to relevant Go module root folders. The file allows a variety of different editors to automatically configure tab width from the config file, leading to the MSCs being displayed consistently on different setups. While some software, like GitLab, had native support for the .editorconfig file, others, like Visual Studio Code, required an extension. This extension requirement was documented in the root readme, and the author visited each developer to ensure they would have the extension installed while answering any questions.

Makefile commands. The team had Makefiles with several commands in place. The team also worked on a handful of Go modules as a part of one microservice, two modules receiving most of the attention. While the two Go modules had equivalent commands, some of them were phrased differently. For example, one Makefile had a target "golangcilint_install" while the other had "lint_install", which were unified to "install_lint". This way it should be easier for developers to remember the target names, as it does not matter on which Go module they are working on.

Targets for running unit and integration tests were also added. This allowed predefining a special build flag required by some of the tests, which a developer would have had to configure themselves to their editor. The required build flag was also documented to the readme in the Go module root.

5.2 Process improvements

Retrospectives. The team was not actively doing retrospectives, but the practice was picked up again based on the author's suggestion to the lead developer. The team maintains an Excel sheet, the rows of which relate to various aspects of the project. As an example, during the retrospectives the team adopted Google's Go style guide and agreed together on deviations from it, preferring CamelCase to Go's capitalisation convention.

5.3 Further work

One participant brought up how moving the developer tooling closer to the developer laptops from external systems could be one point of improvement, to make the automatic feedback more immediate.

Another mentioned how it would be nice if “someone would tell how this [SonarQube] should be read. Of course, there are these numbers here and one can see what has happened, but is there something else in this view that I could benefit from? This may be a little bit less familiar to me”. But they also continued on how a precondition for this would be getting the SonarQube to work consistently, since, as of writing it only shows the most recent run and occasionally causes failures when the branch of the previous run does not match the current one. The failures are seemingly due to the lack of multi-branch support in the SonarQube instance being used.

A third participant suggested how there could be a display in the team’s physical working space for the nightly smoke tests the team has spent time writing. “In practise, no one is [currently] looking at them, they should be checked every morning. And with what probability would it [checking the smoke tests every morning] continue to happen when after the first week interest has been lost at watching them. They could be—or at least in my opinion should be—visible on a display so that the first person arriving at work would see ‘oh, the tests are in the red’. And if they could not solve the problem immediately, they would say to the next arrival ‘hey look at these, they’re red’. And even if no one could do anything then by midday everyone would be looking at them being red, leading to the issues getting fixed. At the moment the tests may be red for a week, and no one notices, or we wonder after a week why they’re broken ‘what happened a week ago?’ And this would only require a display with a status”. Nevertheless, the participant continued that “even if everything has not been polished to perfection, things still work”.

6 Discussion

For a thesis written in the industry, there was a considerable focus on research namely the DX framework (Greiler et al., 2023). Practical tasks were also carried out, but instead of being able to focus on one larger tool or workflow, the tasks were mostly focused on a program code level. This is natural for a job titled ‘software developer’. These levels are illustrated in Figure 11 below.

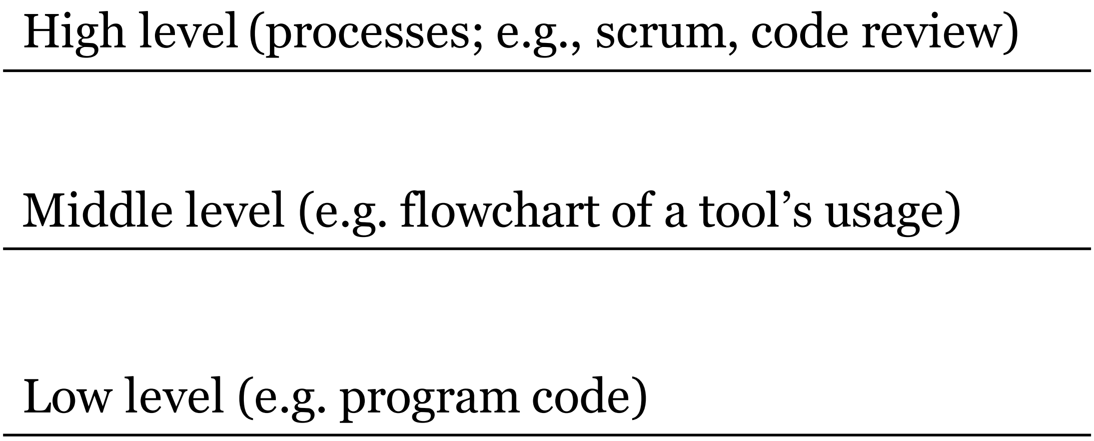


Figure 11. Different levels at which developer experience can be improved.

The most interesting improvements thesis-wise would probably be found at the middle level, where highly generalisable results with for example Design Science Research (DSR) method could be achieved. Higher level changes may be out of scope for the job description of a developer, depending on the organisation and the team.

Nevertheless, focusing on the framework still yielded interesting findings. For instance, one thing that could have been done differently would have been having the participants use their own computer in the interviews, to observe their workflows in more familiar settings. They might have customised or at least configured in a different way compared to the interview laptop. One participant noted how the different colour theme threw them off a bit as everything looked somewhat unfamiliar. While relevant bookmarks to the different systems such as Jenkins and SonarQube were provided, they do not necessarily match the participant’s setup or usual workflows. This aligns well with the mention by Greiler et al. (2023) where developer experience is highly personal. In other words, while UX research may focus on more standardised products with pre-defined workflows, developers being highly adept at using their computer, may have more elaborate or customised ways of interacting with it.

The interviewer’s laptop was used due to the recording software used, but in hindsight recording via MS Teams screen sharing could have worked well. Of course, more time would have been required to test the recording setup with the participant, so there are trade-offs.

Another observation from the factor discussion section 4.2 was how the participants had a tendency to relate DX factors to the organisation, team, or themselves as an individual. Good examples of such factors include *having aligned values* and *psychological safety*. The three levels are illustrated in Figure 12. When one wants actionability out of the interviews, it may be worthwhile to prompt the interviewees specifically about how they relate a factor's importance in terms of their team, rather than the organisation, which can be the most rigid out of the three, or individual preferences as it tends to lead to somewhat banal answers.

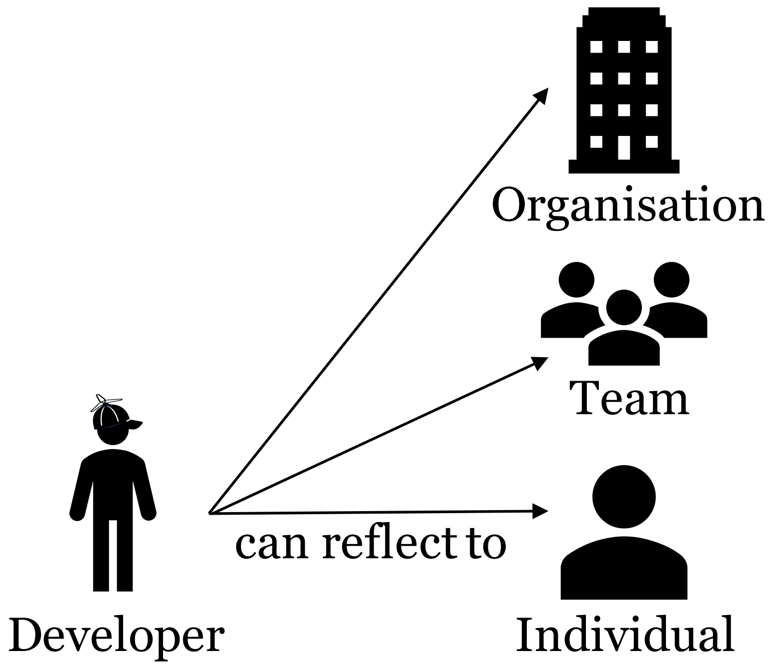


Figure 12. Developer experience factor reflection targets.

Continuing with the theme of specific prompts, at times the participants were confused about whether they were asked to assess the current level of a factor, or how they perceived the importance a factor. As an example, a participant could be asked to rate how their team's *code review process* is or as how important they perceive a *code review process* in general. In the case of this thesis, the participants were always instructed to do the latter. These two alternatives are illustrated in Figure 13.

Both of these ways are still relative to the developer in question. In the interviews one participant described the codebase the case team is working on as small, because they had previously worked on a bigger one. At the same time another participant described the same codebase as wide, stating how when they started in the team there was "a lot of prior work to be understood".

While there is overlap between the different levels in Figure 13 interview-wise, it may be interesting to present it in full (or a comparable taxonomy) and then specifically prompt about the team level. For example, at a large company the organisational environment may require great effort to change, which makes it a relatively *in*actionable target. Similarly, changes based on individual preferences may clash with the individual preferences of others also leading to inaction or even conflict. The hope is that by prompting for the team level, while being aware of the other levels, individuals produce change proposals targeting relatively actionable ground.

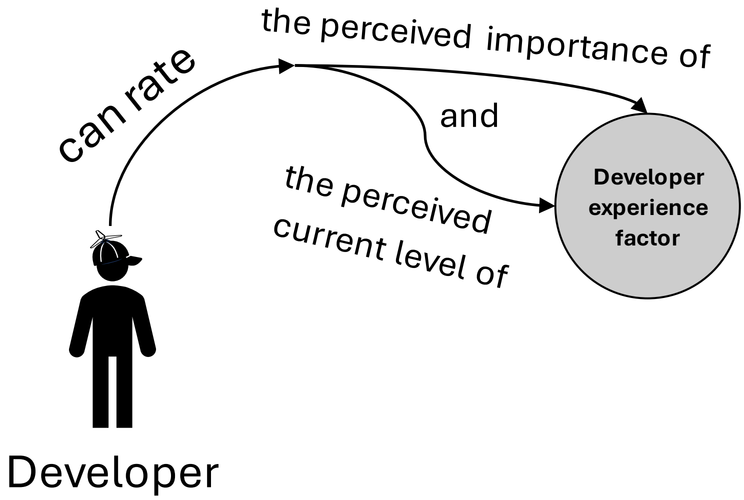


Figure 13. Ways of relating to a developer experience factor.

Some participants expressed interest in rating how well the team is doing in regard to a factor. By combining the perceived importance with the perceived current state of a factor, a more actionable framework could be formulated as a way to help a team to improve their current ways of working. A simple example to utilise these two dimensions for actionability is presented in Figure 14.

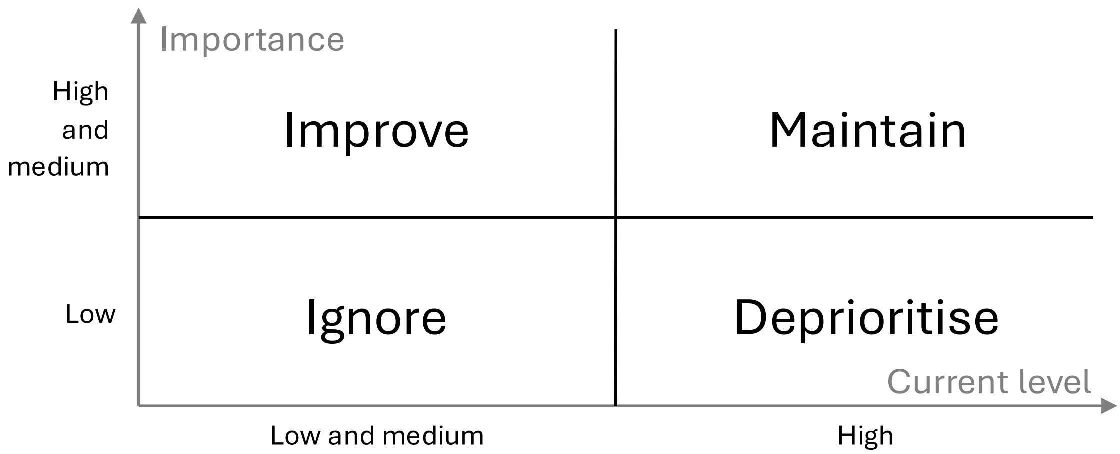


Figure 14. Using importance and current level for actionability.

6.1 Limitations

This thesis focused mostly on the DX framework (Greiler et al., 2023) through qualitative interviews with six software developers. The sample size alone limits the applicability of the results elsewhere, as a large portion of the samples could simply be outliers. Reviewing more literature could also increase the reliability of the results. Despite that, the main contributions in Figures 13 and 14 are likely valid, although reviewing more literature with search terms like 'change management' or 'change facilitation' combined with the themes of the developer experience factors could help hone the exact phrasing further.

One issue discovered in the analysis was how connections between the factors seemed to be formed mostly to the already discussed factors, as can be seen from Table 4 below. The order of DX factors can be seen from Appendix B, categories were interviewed about in numerical order, from top to bottom. In other words, the expectation was that the number of ✓s and ✗s in Table 4 would be roughly equal, whereas now it seems that there is a bias to connect factors to ones that have been discussed before.

Arrow	Source	Target	Points to previous?
A	Making progress without obstacles	Challenging/stimulating work	✓
B	Challenging/stimulating work	Learning	✗
C	Clear goals, scope, requirements	Collaboration between departments	✓
D	Clear goals, scope, requirements	Code review process	✓
E	Having aligned values	Code review process	✓
F	Collaboration between departments	Knowledge sharing	✓
G	Collaboration between departments	Communication	✗
H	Communication	Collaboration between departments	✓
I	Autonomy	Knowledge sharing	✓
J	Communication	Knowledge sharing	✓
K	Communication	Supportiveness	✓
L	Communication	Psychological safety	✓
M	Collaboration between departments	Providing value to the business	✗
N	Supportiveness	Providing value to the business	✗

Table 4. 71% of the time connections were made to previous factors.

Another limitation is that the level at which the participants were instructed to reflect the factors on was not specified strictly. If participants asked for further guidance, they were instructed to focus on themselves "how does the factor affect you", but this did not happen with every participant so the interview results may be somewhat inconsistent.

6.2 Further research

Further research into the developer experience framework (Greiler et al., 2023) could attempt to avoid the aforementioned inconsistency between interviews by utilising what is proposed alongside Figures 12 and 13.

Another thesis could attempt to utilise the proposals from this one in practice. Reviewing change management, meeting facilitation, and more developer experience literature may help in increasing rigor.

One could also try to minimise the leading questions on factors such as *unreasonable deadlines*. But whether this causes more issues by deviating from the original framework is an issue on its own. An interesting parallel to familiarise oneself with before modifying the developer experience framework may be how the System Usability Scale, SUS (Brooke, 1995), has evolved (Lewis, 2018).

Additionally, if the connections between the factors are of interest, based on Table 4 it may be helpful to play with the factor interview order.

The leadership of the case team would have been interested in improving the collaboration between the case team and an offsite one responsible for the other parts of the product and testing. The other site was scoped out to keep the thesis manageable, but similar situations could be fertile ground for further research.

7 Conclusions

This thesis starts with two research questions, **RQ1** "how do the software developers in the case team perceive the developer experience factors from the framework?" and **RQ2** "where do practitioners direct their attention when they want to improve developer experience?".

In terms of **RQ1**, the thesis found ways to prompt interviewees more specifically about the DX factors by observing how some of the differing importance ratings were explainable by one participant reflecting the factor on the company and another on the team. For example, *having aligned values* was rated as low importance when the participant was thinking about how well the company's values match theirs, but as high importance when the participant was thinking how it may speed up the *code review process* in the team. These different reflection targets are illustrated in Figure 12. Additionally, during the interviews some developers expressed interest in rating the perceived current level of a DX factor, in addition to just the perceived importance. This is illustrated in Figure 13.

Chapter 5 focuses on answering **RQ2**. The attention seems to have landed on improving dependency caching, various aspects of the test suite (speed/reliability, coverage), IDE configuration, logging, and developer tooling. In other words, attention was mostly directed towards relatively low-level aspects, see Figure 11, of a developer's work environment, which is only a subset of DX as defined by the various taxonomies presented in Figure 2.

In pursuit of improving developer experience, utilising specific prompting à la Figure 12 and 13, the two dimensions of perceived importance and current level of a DX factor can be combined to produce more direct actionability, as Figure 14 shows.

References

Brooke, J. (1995). SUS: A quick and dirty usability scale. *Usability Eval. Ind.*, *189*.

Davis, G. (2023, June 8). *Developer experience: What is it and why should you care?* The GitHub Blog. https://github.blog/2023-06-08-developer-experience-what-is-it-and-why-should-you-care/

DiCicco-Bloom, B., & Crabtree, B. F. (2006). The qualitative research interview. *Medical Education*, *40*(4), 314–321. https://doi.org/10.1111/j.1365-2929.2006.02418.x

EditorConfig. (n.d.). Retrieved May 16, 2024, from https://editorconfig.org/

Evans, E. (2004). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley.

Fagerholm, F., & Münch, J. (2012). Developer experience: Concept and definition. *2012 International Conference on Software and System Process (ICSSP)*, 73–77. https://ieeexplore.ieee.org/document/6225984

Fowler, M. (2014, January 15). *Bounded Context*. Martinfowler.Com. https://martinfowler.com/bliki/BoundedContext.html. Retrieved 11.5.2024.

Greiler, M., & Noda, A. (2022, April 5). *DX framework*. https://github.com/get-dx/dx-framework. Retrieved 12.5.2024.

Greiler, M., Storey, M.-A., & Noda, A. (2023). An Actionable Framework for Understanding and Improving Developer Experience. *IEEE Transactions on Software Engineering*, *49*(4), 1411–1425. IEEE Transactions on Software Engineering.

Kortesuo, K. (2013, January 11). *Osmo A. Wiion lait latistavat viestinnän*. Ei oo totta. https://eioototta.fi/osmo-wiion-lait-latistavat-viestinnan/. Retrieved 13.5.2024.

Koster, R. (2004). *Theory of Fun for Game Design*. Paraglyph, Incorporated. https://archive.org/details/theoryoffunforgamedesign2ndeditionpdfbo/mode/2up

Lewis, J. R. (2018). The System Usability Scale: Past, Present, and Future. *International Journal of Human-Computer Interaction*, *34*(7), 577–590.

Matthews, G., & Zeidner, M. (2004). *Traits, States, and the Trilogy of Mind: An Adaptive Perspective on Intellectual Functioning*. Routledge. https://doi.org/10.4324/9781410610515-14

Meier, R. (2015, June 30). Why Do We Pay These People Anyway? *Google Developers*. https://medium.com/google-developers/why-do-we-pay-these-people-anyway-d7ed706d6d55. Retrieved 22.2.2024.

Mounts. (n.d.). Docker Documentation. Retrieved May 17, 2024, from https://docs.docker.com/build/guide/mounts/

Nylund, A. (2020). *A multivocal literature review on developer experience*. https://aaltodoc.aalto.fi/handle/123456789/43567

Osmo Antero Wiio. (1978). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Osmo_Antero_Wiio&oldid=1195702618. Retrieved 13.5.2024.

Siponen, M., & Baskerville, R. (2018). Intervention Effect Rates as a Path to Research Relevance: Information Systems Security Example. *Journal of the Association for Information Systems*, *19*(4), 247–265.

Wolpert, D. H., & Macready, W. G. (2005). Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, *9*(6), 721–735. https://doi.org/10.1109/TEVC.2005.856205

A Interview guideline

45-minute session

First page ~10 minutes. Time allocations are flexible and include spare.

People to interview:

- Software developers in the team, also new ones.

Disclaimer:

- Ask for permission to record (screen & audio) + tell them that the interview is used for research and that their data will be looked at by a researcher but will be anonymized before ever being shared publicly.

Demographic questions (sent beforehand via email):

- Current role and responsibilities.

The goal of this research:

In this interview, the goal is to understand what aspects of our development infrastructure affect a goal of trying to achieve 90% test coverage in a recent (within 1–2 weeks) merge request of yours. This will ground the thesis within our team's context. A goal of the research is to understand how continuous integration pipelines affect developer experience, especially productivity and impact-wise. Here productivity means "how quickly or simply a change can be made to a codebase" and impact "how frictionless it is to move idea to production". Additionally, there is satisfaction, i.e. "how the environment, workflows, and tools affect developer happiness".

5 minutes for explaining the goal of this research.

Unguided impressions:

What are the first things that immediately come to your mind when you think about your experiences of developing software in this team?

- What affects you?
- What affects our team?

10 minutes for unguided impressions.

Part 2

"Now that we've had some warm-up, I'd like to introduce to you developer experience factors from a research paper this thesis is based on." ~25 minutes

DX framework factors:

- Show the interviewee the developer experience factors presented in the final paper by Greiler et al. (copied below). Present them with the name of the factor + a short description of what each factor means.
- Ask them about the perceived importance of each factor (low, medium, or high)
- For factors they rated as **low**, ask them "what led you to this one?".
- For factors they rated as **high**, ask them "what led you to this one?".
- Can you think of additional factors that are missing from this list or that we have not talked about that are important or significantly influence your developer experience?

Final Thoughts (to catch gaps):

Is there anything that you feel is missing that is important for this topic, that we did not talk about? (~5 minutes)

B Developer experience factors and categories

1. Development and release ● Codebase health ● Development environment ● Automated testing ● Frictionless release	2. Product management ● Clear goals, scope, requirements ● Working iteratively (small WIPs) ● Unreasonable deadlines ● Having a say on roadmap/priorities ● Providing value to the business
3. Collaboration and culture ● Supportiveness ● Knowledge sharing ● Feeling connected ● Code review process ● Collaboration between departments ● Psychological safety ● Communication ● Having aligned values ● Getting recognition	4. Developer flow and fulfilment ● Autonomy ● Challenging/stimulating work ● Making progress without obstacles ● Uninterrupted time ● Work-life balance ● Learning ● Stability of job and team ● Clear paths for career growth