Mob Programming: From Avant-Garde Experimentation to Established Practice

Daniel Ståhl[a,*], Torvald Mårtensson[b]

[a] *Ericsson AB, Linköping, Sweden; Linköping University, Linköping, Sweden*
[b] *Saab AB, Linköping, Sweden*

Abstract

Mob programming is an emergent practice in the industry, attracting increasing attention from the research community. Superficially a simple concept — similar to pair programming, but with more people — many of the same concerns arise as with pair programming, only more so: can it truly be efficient, is it for everybody, and what are the benefits and risks involved? In this paper we show that while research interest in this topic is increasing, much of the published literature to date constitutes experience reports, with a number of implicit differences in how mob programming is practiced. To transform mob programming from an area of avant-garde experimentation into a well understood practice in the toolbox of mainstream software engineering, such implicit differences need to be exposed, analyzed and documented, and the contextual enabling factors must be investigated. In this paper we take an important step in that direction by identifying variations in the practice and providing concrete guidance for creating an organizational and social environment where mob programming can be practiced both effectively and safely.

Keywords: mob programming, mobbing, agile, pair programming, exploratory testing, research roadmap

1. Introduction

Mob programming was first introduced in literature by Hohman and Slocum (2001), to later be elaborated and popularized by Zuill and Meadows (2014) who described the success experienced after more or less serendipitously developing the practice in their own team. At its simplest, mob programming can be understood as a form of pair programming, but with three or more participants. In the description given by Zuill and Meadows, the practice is formalized with dedicated roles, such as navigator and driver, that continually rotate among the mob programming participants and fulfill different needs in the programming process. In this sense there is considerable conceptual overlap between mob programming and the idea of strong pairing (Falco, 2014), which emphasizes that every idea must pass through another person's brain to enter into the code. The practice also has similarities to Randori coding dojos (Rooksby et al., 2014) — with designated driver, navigator and "audience" — with the difference that mob programming is intended for actual software development, as opposed to being designed for learning purposes.

As researchers, we have approached mob programming from our experiences with exploratory testing (Bach, 2003; Hendrickson, 2013), applied collaboratively in large-scale software development projects (Mårtensson et al., 2019, 2017). In this work we have observed remarkable value created and insights gained, particularly when individuals of different backgrounds and competences are brought together to jointly run a single testing scenario. This may seem inefficient — surely the precious time of these senior engineers would be better spent if they each ran their own scenario in parallel? In fact, the discussions, learning and rapid troubleshooting that resulted from the shared focus on a single task not only let them bring ready-to-implement solutions back to their respective organizations, but also led to tangible joy and enthusiasm at actually working as a team rather than as a collection of individuals.

*Corresponding author. Tel.: +46 76 14 97 573.
Email addresses: daniel.stahl@ericsson.com; daniel.stahl@liu.se (Daniel Ståhl), torvald.martensson@saabgroup.com (Torvald Mårtensson)

These positive experiences of collaborative exploratory testing raises the question: what would an analogous practice for development — as opposed to testing — look like? It is from this point of view that we approach mob programming, as such an analogous practice, prompting us to investigate it further. Recognizing that this practice, as any other, may have both positive and negative effects and that it may be applied in different ways, in varying contexts and for a wide array of types of tasks, this study seeks to answer the following research question: *When and how is mob programming an effective and efficient method of software development?* To address this question, we take a dual approach. First, experiences and research on the practice in published literature are investigated. Second, perceptions are gathered from current industry practitioners. From this, the variants, benefits and risks of the practice are then discussed and a research roadmap to further explain the practice are proposed.

The contribution of this paper is three-fold. First, it provides an overview of the state of research for an emergent practice in the industry. Second, it offers guidance to software engineers to help them evaluate in a more systematic way whether mob programming may be feasible for them, and supports them in bootstrapping the practice by providing relevant and actionable recommendations based on published literature and industry experiences. Third, it proposes a roadmap of subsequent research on mob programming.

The remainder of this paper is structured as follows. Section 2 provides an overview of the research method, while Section 3 surveys related work. In Section 4, results from interviews with mob programming practitioners are presented. The interview data is then analyzed and discussed in Section 5 and a research roadmap is proposed in Section 6. Finally, threats to validity and reliability are discussed and the paper is concluded in Sections 7 and 8, respectively.

2. Research Method

To answer the research question of *When and how is mob programming an effective and efficient method of software development?*, a step-wise research method was designed.

2.1. Overview

A high level overview of this research method is provided in Figure 1:

- Related work was studied to identify salient factors — claimed benefits, risks and variables — in mob programming methodology and group them into Areas of Interest relevant to mob programming. This step is presented in Section 3.
- Engineers already practicing mob programming were interviewed based on these Areas of Interest. This step is presented in Section 4.
- The interview data was then analyzed from the perspective of each of the Areas of Interest, with organizational and social implications considered. This is discussed in Section 5.
- Based on the analysis, a research roadmap was designed to address what we consider the most salient open questions in mob programming. This roadmap proposal is presented in Section 6.

The methodology of each respective step is described in detail below.

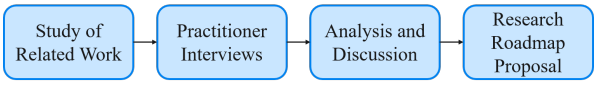

Figure 1: Research process overview.

2.2. Study of Related Work

In order to form an understanding of the current state of the art in mob programming research, a systematic review of published mob programming literature was conducted, according to the guidelines established by Kitchenham (2004). A literature review protocol was designed, containing the question driving the review ("Which benefits, risks and/or variables of mob programming are claimed and/or hypothesized in literature?") along with inclusion and exclusion criteria (see Table 1). Following Kitchenham's guidelines, the stages of the review were as follows:

- **Identification of research:** Iterative trial searches to arrive at a conducive search string. To cast a wide net, the search strings were kept

short and simple as to not exclude any sources that might not explicitly address the question of benefits, risks and variables, but rather provide e.g. an experience report.

- **Selection of primary studies:** Exclusion of duplicates and conference reviews and other items lacking available content beyond abstract and/or references (EC_{1-3} in Table 1).
- **Study quality assessment:** The relevance of the remaining publications was assessed in a first review of each paper, and papers deemed irrelevant to the review question were excluded (EC_4 in Table 1).
- **Data extraction and monitoring:** The remaining sources were read in full and are discussed in Section 3. All claims or hypotheses regarding mob programming factors — potential benefits (e.g. mob programming leading to improved software quality), risks (e.g. mob programming causing exhaustion) and variables (e.g. role rotation intervals) — were retrieved. From the resulting list, pre-defined selection criteria were applied to identify key factors to group into Areas of Interest and focus on in subsequent steps of the study.

To achieve consistency during the review, all publications were reviewed by the same researcher, whereupon the analysis was reviewed by the other researcher in a two-step processes in order to secure quality and correctness.

The output of the study of related work was a set of mob programming Areas of Interest to focus on in practitioner interviews (see Section 2.3).

2.3. Practitioner Interviews

Following the study of related work (see Section 2.2) individual interviews were conducted with 8 engineers already experienced in mob programming. These engineers constitute one of several teams responsible for the development, support and maintenance of a simulator product in a large-scale systems development company. The team comprised both male and female developers, with a high degree of age diversity, spanning from 27 to 59 years old. In particular, their experience from developing this type of simulator varied, with some of the team members having participated in the development of multiple previous generations over the preceding 15 years, and others having no such experience prior to joining the team and starting mob programming. The team would run a single continuous mob session throughout the day, every day, in which team members would participate to varying degrees and at different times according to their preferences and tasks.

The interviews were semi-structured and conducted individually by one researcher, either in person or remotely, and in Swedish or English according to the preferences of each interviewee. The interview guide (presented in Section 4) consisted of behavior/experience questions, as described by Hove and Anda (2005), corresponding to the retrieved Areas of Interest identified in Section 3. For the variables, reflexive questions were also included to encourage the interviewees to speculate on alternative ways of working, based on their personal experiences. The interviewees were provided a copy of the questions at least 24 hours in advance to afford them the opportunity to prepare. The transcribed responses were read back to the interviewees during the interviews to ensure correct interpretation, and they were explicitly encouraged to provide spontaneous reflections and/or elaborations.

The interview results were thematically coded and mapped according to the thematic coding analysis guidelines presented by Robson and McCartan (2016):

- **Familiarizing with the data:** The transcripts were read and re-read, with initial ideas noted down.
- **Generating initial codes:** Particular statements of interest across the entire data set were identified and systematically coded.
- **Identifying themes:** The codes were grouped into candidate themes, which were then checked against codes and other themes to ensure accuracy and avoid overlap. During this process the themes and codes were iteratively revised and refined.
- **Integration and interpretation:** The data was structured in a thematic map to support subsequent analysis, and the quality of the map was re-evaluated.

This thematic coding analysis was conducted iteratively to increase the quality of the analysis, resulting in a thematic map of statements and reflections made by the practitioners. To mitigate

any biases on behalf of any one of the individual researchers, it was carried out by each of them in parallel, whereupon the results were compared, discussed and merged into a single thematic map. This map is presented in detail in Section 4.

2.4. Analysis and Discussion

In the analysis, the entire thematic map resulting from the practitioner interviews was studied from the perspective of each of the Areas of Interest derived from related work (see Section 2.2). This mapping resulted in a matrix of themes in the interview results and the Areas of Interest, enabling a structured and exhaustive analysis. This analysis led to new insights and also revealed multiple open questions, as discussed in detail in Section 5.

2.5. Research Roadmap Proposal

The analysis (see Section 2.4) provided not only answers, but also left several salient questions remaining. In an emergent field, such as mob programming, this is to be expected: we argue that an important part of the research process at this stage is to map the field and identify relevant questions to investigate. Consequently, to encourage and facilitate further work, the open questions revealed in the analysis were categorized as pertaining to one of four research topics: organizational enablement of mob programming, its social dynamics, quantification of the practice and its applicability to a remote work paradigm.

3. Study of Related Work

To identify potential risks, benefits and variables in mob programming practice, informing subsequent steps in the research process (see Section 2), a study of related work was conducted. To identify sources, the inclusion and exclusion criteria listed in Table 1 were used.

As shown in Table 1, the initial search in four databases yielded a total of 16 unique results. These were then culled through subsequent application of exclusion criteria to 11 papers:

- $\mathbf{P}_1$: Mob Programming: A Systematic Literature Review (Shiraishi et al., 2019)
- $\mathbf{P}_2$: An Intelligent-Agent Facilitated Scaffold for Fostering Reflection in a Team-Based Project Course (Sankaranarayanan et al., 2019)
- $\mathbf{P}_3$: Leveraging the Mob Mentality: An Experience Report on Mob Programming (Buchan and Pearl, 2018)
- $\mathbf{P}_4$: Mob Programming: The State of the Art and Three Case Studies of Open Source Software (Kattan et al., 2017)
- $\mathbf{P}_5$: Swarm or Pair? Strengths and Weaknesses of Pair Programming and Mob Programming (Kattan et al., 2018)
- $\mathbf{P}_6$: Theory of Altruism on Software Development Practices Patterns (Kattan, 2018)
- $\mathbf{P}_7$: Mob Programming — A Promising Innovation in the Agile Toolkit (Balijepally et al., 2017)
- $\mathbf{P}_8$: From Pair Programming to Mob Programming to Mob Architecting (Lilienthal, 2017)
- $\mathbf{P}_9$: Software Development Practices Patterns (Kattan and Goldman, 2017)
- $\mathbf{P}_{10}$: Mob Programming: Find Fun Faster (Boekhout, 2016)
- $\mathbf{P}_{11}$: Mob Programming — What Works, What Doesn't (Wilson, 2015)

It is noteworthy that none of the sources are journal papers, and most of them are relatively short (the majority being six pages or shorter) and recent (the majority being published 2018 or later). Apart from the sources listed above, we identify two non-peer reviewed but noteworthy books by Zuill and Meadows (2016) and Pearl (2018), respectively, which we refer to in relation to our study findings in subsequent sections.

Four of the publications present literature reviews or overviews of varying degrees of formality and for different purposes (Balijepally et al., 2017; Kattan et al., 2017, 2018; Shiraishi et al., 2019). Five publications present experience reports, but we note that only three of these (Boekhout, 2016; Buchan and Pearl, 2018; Wilson, 2015) are from an industry setting, with the remaining two being reports from educational contexts (Kattan et al., 2017; Sankaranarayanan et al., 2019). In addition, Buchan and Pearl (2018) presents an online survey of mob programming practitioners. Meanwhile, two publications propose planned research projects (Kattan, 2018; Kattan and Goldman, 2017) while

Inclusion Criteria		Yield
IC_1	Scopus search on 2020-03-06 for `TITLE-ABS-KEY("mob programming") AND SUBJAREA(COMP)`	15
IC_2	Web of Science search on 2020-03-06 for `TS=("mob programming") AND SU=Computer Science`	6
IC_3	IEEE Xplore search on 2020-03-06 for `("mob programming")`	2
IC_4	arXiv.org search on 2020-03-06 for `title="mob programming"; OR abstract="mob programming"; OR comments="mob programming"`	1
Exclusion Criteria		**Remaining Set**
EC_1	Union of results, excluding duplicates (in which case the most recent publication is used) but not extensions	16
EC_2	Exclusion of conference reviews, posters, workshop proceedings, or other items lacking available content beyond abstract and/or references	13
EC_3	Exclusion of publications unavailable to the authors in English	12
EC_4	Exclusion of publications that, through review of abstracts, are shown not to address the software engineering practice of mob programming	11

Table 1: Inclusion and exclusion criteria of the systematic literature review, along with the number of papers yielded from the performed searches and remaining after application of exclusion criteria, respectively.

Lilienthal (2017) suggests the further evolution of mob programming towards mob architecting.

Overall we regard these findings as indicative of an immature but emerging research field.

3.1. Literature Reviews

As noted above, the studied set of publications contained four literature reviews, the most recent and comprehensive of these being P_1 (Shiraishi et al., 2019). Using Scopus and Google Scholar, P_1 found 108 papers in their initial search, which was then culled to only 10 by judging the remainder irrelevant[1]. We note that with the exception of a Bachelor's thesis (Aune et al., 2018) and the early work by Zuill and Meadows (2014) and Hohman and Slocum (2001) (discussed in Section 1), all publications in P_1 are included in our study.

The purpose of the literature review in P_1 partially overlapped with our work and identified a set of reported benefits, risks and variables. Rather than copy the findings of P_1, we decided to mine the identified primary sources and then use the findings presented in P_1 as verification of our own results (see Section 3.2).

3.2. Empirical Sources

For the purposes of this study, we collected from sources P_{1-11} empirically observed or hypothesized factors — benefits, risks and/or variables — of mob programming that may serve to inform subsequent steps of our research process (see Section 2). These identified factors, along with source(s), are listed in Table 2. Factors indirectly claimed by referencing other sources included in the literature review were omitted. Similarly, claims regarding e.g. placement of screens or type of chairs used were omitted, since we consider them sub-factors — presumably dependent on the personal preferences of individuals — contributing to the physical comfort of the team members (included as factor F_{R3} in the table). Finally, while several sources discuss risks *to* mob programming practice (e.g. certain organizational cultures impeding its introduction), for the purposes of this research we have focused on risks *of* mob programming practice (e.g. reduced productivity).

As can be seen from Table 2, we found tangible claimed or hypothesized factors in far from all

[1]We note that the large number of papers is caused by numerous irrelevant hits returned by Google Scholar.

Benefits		Source(s)
F_{B1}	Effective real time use of the team's collective intelligence	P_3, P_8
F_{B2}	Lead time optimization	P_3, P_8
F_{B3}	Improved productivity with less context switching and distractions	P_3, P_7
F_{B4}	Increased team ownership, engagement and consistency in design	P_3, P_{10}
F_{B5}	Increased consistency and effectiveness in tooling usage	P_3, P_{10}
F_{B6}	Accelerated learning	P_{3-4}, P_{7-8}, P_{10-11}
F_{B7}	Improved predictability and better distribution of work	P_3
F_{B8}	Better informed design decisions	P_3
F_{B9}	Increased team confidence leading to bolder behavior	P_3, P_{11}
F_{B10}	Improved developer satisfaction and morale	P_3, P_7
F_{B11}	Strengthened interpersonal relationships	P_4, P_{11}
F_{B12}	Improved software quality and reduced technical debt	P_{7-8}
F_{B13}	Increased team self-sufficiency	P_{10}
Variables		Source(s)
F_{V1}	The type of task selected for mobbing sessions	P_{3-4}, P_8, P_{10-11}
F_{V2}	Using shared or individual equipment	P_3, P_4
F_{V3}	Driver rotation scheme	P_{3-4}, P_{7-8}, P_{10-11}
F_{V4}	Participation of external specialists or stakeholders	P_3, P_{11}
F_{V5}	Predominance of experienced or non-experienced drivers	P_3
F_{V6}	Allowing individual members to fluidly break off and rejoin	P_3, P_{11}
F_{V7}	Size of the mob	P_3, P_{11}
F_{V8}	Length and frequency of mobbing sessions	P_3, P_{10-11}
F_{V9}	Break scheme	P_{3-4}, P_{10-11}
F_{V10}	Retrospectives scheme	P_4, P_{10}
F_{V11}	Rotation of members between teams	P_{11}
Risks		Source(s)
F_{R1}	Reduced productivity	P_3, P_{10}
F_{R2}	Interpersonal challenges	P_3
F_{R3}	Physical discomfort or exhaustion	P_{3-4}, P_7
F_{R4}	Groupthink or unquestioning deference to dominant team members	P_4, P_{11}

Table 2: Mob programming factors — benefits, risks and relevant variables — claimed in literature.

of the reviewed sources. Indeed, several of the reviewed papers merely relate claims made by other sources — highlighting the need for further empirical research in this field, as identified by Balijepally et al. (2017). It is also worth noting that several of the variables (F_{V1-V11}) listed in Table 2 are implicit — in other words, they are not addressed directly in the studied paper, but only become apparent through comparison each paper's respective description of the practice.

Comparing the identified set of benefits, risks and variables with that of P_1 (see Section 3.1) we find

that all factors in that earlier study are represented in Table 2, with the exception of certain risks *to* mob programming, as opposed to *of* mob programming (as discussed above).

To identify benefits and risks for further investigation in this study, two criteria were used. First, in order to cover topics of demonstrably high interest in related work, any factors with more than three sources discussing them were selected. Second, to cover areas of potential controversy, claims or hypotheses put forward by two or more sources contradicting similarly supported claims or hypotheses were added. Through application of these two criteria, Areas of Interest AoI_{1-4} listed in Table 3 were selected for further investigation.

Given that variables (F_{V1-V11}) were not necessarily explicit in the studied papers, they were included for either fulfilling the criteria above, or being described differently in two or more sources. This process produced two additional Areas of Interest (AoI_{5-6}).

4. Practitioner Experiences

This section presents the results from the interviews with mob programming practitioners. To capture the insights of developers already experienced in mob programming, the members of a development team having practiced it on a daily basis for nine months were interviewed (see Section 2.3). The interview guide comprised 13 questions designed to address the Areas of Interest retrieved from literature (see Table 3), with each Area of Interest covered by at least one question.

The thematic map — showing only the top level themes — constructed from the interview results is shown in Figure 2. As evident in the figure and as expected from a thematic coding analysis, the derived themes do not map one-to-one to the interview questions. Rather, the map reflects how new topics were spontaneously raised by the interviewees or how they in their responses made connections between topics considered separate when constructing the guide. These fourteen themes are discussed in four subsections below:

- Outcomes (**Productivity**, **Lead Time Optimization**, **Suitability**, **Learning**, **Quality**)
- Practice Application (**Breaks**, **Roles**, **Driver Rotation**, **Remote**)

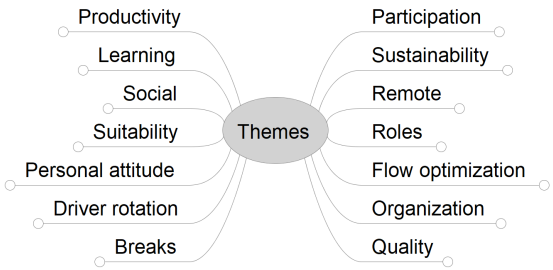

Figure 2: Thematic map of interviews with experienced practitioners.

- Interpersonal Context (**Social**, **Organization**)
- Individual Factors (**Participation**, **Personal Attitude**, **Sustainability**)

4.1. Outcomes

With regards to **Productivity**, all of the interviewees found that the overall productivity of the team was either "at about the same level as before" or slightly higher, with several of them noting that there were both positive and negative effects and that it would depend on the nature of the work (discussed in detail below). On the negative side, two remarked on the absence of "flow" while mobbing: "You don't enter into a flow like you do working on your own. I have never experienced that in a mob." The majority of the interviewees also remarked on the importance of the mob members' individual abilities. On the one hand, for a senior developer with in-depth knowledge of the problem domain "it can feel like [the mob] is slower [than working alone]". On the other hand, a major factor in the mob's productivity is that there is always "someone who knows the domain", but when those senior developers didn't participate that advantage was lost. As one interviewee put it, "it is as if you raise the below-average and pull down the above-average [developers]". Apart from any immediate effects on productivity, two of the interviewees speculated that mobbing should lead to long term improvements from better quality and/or from increased learning.

Related to productivity, there was consensus that mob programming contributed to **Lead Time Optimization**, for several reasons. First and foremost the mob could only do one thing at a time, but did it fast and did it well and truly done —

Area of Interest		Description	Included Factors
AoI_1	Productivity	Reduced productivity is arguably the most intuitive risk of mob programming and is indeed discussed by two sources. At the same time, potentially increased productivity is claimed by two sources.	F_{B3}, F_{R1}
AoI_2	Learning	With more references (six) than any other potential benefit, the ability of mob programming to improve learning was selected. We find this factor particularly interesting due the way e.g. Hendrickson (2013) describes exploratory testing as being focused on learning (see Section 1 for reasoning around mob programming as analogous to the type of collaborative exploratory testing we have studied in previous work).	F_{B6}
AoI_3	Discomfort and/or exhaustion	Three sources discuss the risk of mob programming team members becoming exhausted unless taking sufficient breaks, and the problem of unsuitable working environments leading to physical discomfort or pain.	F_{R3}
AoI_4	Social dynamics	While two sources claim strengthened relationships, increased trust and better team spirit, one warns of mob programming placing great strain on relationships with high risk of tension and conflict.	F_{B11}, F_{R2}
AoI_5	Task suitability and mob composition	From the experience reports, the more challenging the development task, the more applicable mob programming appears to be. This is in line with findings on the efficacy of pair programming (Dybå et al., 2007; Sun et al., 2015). This is again analogous to experiences from exploratory testing: Gregory and Crispin (2014) explain how exploratory testing shouldn't be used for "the day-to-day repetitive regression testing". Related to this is the question of what the optimal composition of the mob is — depending on what one seeks to optimize for.	F_{V1}, F_{V4}, F_{V5}
AoI_6	Role rotations and breaks	It is generally agreed that it is important to take breaks, to rotate the roles and take turns, but there is little consensus on how that rotation best happens (e.g. fixed rotation schemes, ad-hoc rotation or test-driven development "ping-pong" switching) or how and when to take breaks (e.g. should breaks be short or long, and should the mob break as one or should individuals come and go as they need?).	F_{V3}, F_{V6}, F_{V8}, F_{V9}, F_{R3}

Table 3: Areas of Interest derived from mob programming factors identified in literature.

as a consequence of having all the needed competences in one place. Keeping that focus in the face of customer support requests can be challenging, however, which led the team to implement a rotating dedicated support responsibility (a "supporting pair") in order to "shield" the mob. Other reasons given for improved lead time optimization were cutting or reducing steps in the process that were no longer needed (e.g. "we don't need the preliminary [investigation] work as much"), eliminating waste in

the form of waiting for input from domain experts ("the mob is never stuck, you are always moving forward") and increased predictability. This perceived increase in predictability was caused by three factors. First, the mob evened out the performance of individual members. Second, the risk of getting stuck was greatly reduced. Third, vulnerability to unplanned absence of key individuals was lessened.

That being said, the **Suitability** of mob programming was viewed as varying depending on the type of work. Three factors influencing its suitability were identified by the interviewees: size, complexity and dependencies. Large tasks require multiple concepts to be kept in mind in parallel, and having more minds on hand clearly helps with this. To quote one of the interviewees, in "the tricky places where there isn't one correct answer" the mob was seen as valuable because of the built-in discussion and back-and-forth of ideas. Finally, any far-reaching system impacts and dependencies of the work were more likely to be caught with more eyes focused on it. Conversely, small, trivial and contained tasks were unanimously regarded as inefficient to handle as a mob. Customer support was seen as unsuitable because of its reactive nature — the mob was felt to work best when focused on a problem over long periods of time rather than quickly reacting to incidents — leading to the "supporting pair" described previously. Additionally, code review was done individually since "everybody wants to browse [the code] in their own way, and with a single screen that's pretty hard to do", but several of the interviewees reported very positive experiences from using the mob for high-level design work and breakdown of tasks: "We have tried mobbing systemization and breakdown of tasks and even had customers participate in that. [...] That has worked really well."

Three of the interviewees found that mobbing was great for **Learning**, e.g. stating that "it is really positive, you learn a lot", "it is impossible to not pick things up" and that "you always have somebody next to you whom you can ask". Interestingly, several others remarked that they thought it was a great way for *others* to learn, though not necessarily for them, while one reported having high expectations but being disappointed. Several remarked that learning in a mob may be fast, but superficial: "you get spoon-fed a lot of information [but] I miss experimenting and searching on my own, learning how and where to find the answers" whereas others felt that joining as a beginner can be challenging, because learning in a mob requires "some basic level of understanding". This may depend on the composition of the mob, however: participating as a junior in a mob of seniors "it is so fast you can't really keep up". That being said, there was a consensus that the mob helped spread a shared view of the system and accelerated the spread of new ideas and practices in the team.

With only two sources from previous work discussing effects on **Quality** (see Section 3.2), this was not included in the interview guide. Even so, no less than four of the interviewees spontaneously remarked on what they perceived as improved quality from mob programming, e.g. stating that "the resulting design is better than what it would have been if you had done it yourself", "working alone you miss things, you forget things [but] that doesn't happen as much when the mob is at work" and "the things we have done as a mob are of better quality and have fewer problems". Unfortunately, the team had no quantitative data to support (or refute) this perception, but "it is a feeling".

These practitioner experiences of mob programming outcomes largely confirm reports in related work, e.g. that mobbing "appeared to improve" productivity and "reduced the number of in-progress work items" (Buchan and Pearl, 2018) and that it should not be universally applied to all types of tasks (Wilson, 2015). Quantitative data is lacking, however: the data presented in primary sources is based on *perceived* effects (by the author, studied teams or survey respondents), rather than objectively measured effects.

4.2. Practice Application

Something that differs between applications of mob programming is whether and how the mob handles **Breaks**. Apart from lunch and one coffee break in the morning and one in the afternoon, the studied team (described in Section 2.3) has not employed any regular break schemered, but would be in session all day, every day (i.e. six or more hours, five days a week, with some members participating nearly all the time and others to a lesser extent, as discussed in Section 4.4). Once again, the interviewees reported very different experiences: while some had no trouble at all, others were left completely exhausted by the long uninterrupted sessions. Several remarked that they would take frequent breaks when working alone, but that they "felt that they get stuck in the mob". While some

experienced the mob as an "eight hour long meeting" requiring intense and unbroken focus, others described how they were able to take short breaks in the privacy of their own minds throughout the mobbing session. Yet in the interviews, nearly all team members said they would like to try more frequent breaks, e.g. to facilitate discussions or "something Pomodoro[2] inspired". This is particularly interesting in light of the recommendation by Zuill and Meadows (2016) to start out with a few hours of mobbing per week, and that the "the Mob team [...] should not be required to scale up unless they are ready", acknowledging that mobbing can be a strain — particularly if one is not used to it. The dilemma, as demonstrated by the interviews, is that the team is not necessarily of one mind or at the same level of readiness, creating a source of potential tension.

The studied team's experiences of assigning **Roles** mostly touched upon the navigator role and its relationship to the driver. The team didn't adhere to any formal method of designating the navigator: "we have one person at the keyboard and everybody else navigating", even though several remarked that "it can get rather confusing". Some wanted to try mob programming more "by the book" with a dedicated navigator. Others claimed that the team had already tried having a rotating navigator role and found that it didn't work well, because "someone might be unable to say what they want to say, or the dedicated navigator doesn't know what we should do". One interviewee speculated that "assigning who should be the navigator might work well" (as opposed to a rotating responsibility). The team also hadn't tried appointing or bringing in any moderator or facilitator.

With regards to **Driver Rotation** the team had experimented with different intervals and eventually settled on a formula of ten minutes minus the number of participants to make it easier to maintain focus in between driving. To exemplify, with five members in the mob the team would rotate driver every five minutes. The team hadn't tried non-timer based methods of rotation, but one of them speculated that a risk of letting anyone start driving "is that someone who gets frustrated navigating takes over the keyboard and does their own thing", remarking that this is a common problem with pair programming. The team had also at times let "someone who clearly should be navigating" skip the driver role (e.g. because they might be most familiar with the problem at hand).

A number of statements concerned **Remote** mob programming. The interviews were carried out in the early months of the covid-19 pandemic, during which the team had transitioned from mob programming in the office to mob programming remotely. At the time of the interviews, the team had practiced daily remote mobbing for approximately two months — in other words, some 30-40 sessions. The general experience was that this "works remarkably well", but once again there were individual differences leading to a change in the constellation of the mob: some who didn't participate as much in the office found it easier to join remote mob sessions, while others had the opposite experience and now participated less. Technological prerequisites were mentioned repeatedly, not least a stable low latency connection to be able to follow along as the driver scrolled in the text editor. As a consequence, the team found that they were unable to use the screen sharing feature of the video conference application, but worked in a concurrent editor on a shared remote host in the company's internal cloud. One issue raised by multiple interviewees was how you can't see the facial expressions of the others unless you would turn on cameras — but displaying all the camera feeds required additional screens and was usually not done. One of the interviewees speculated that turning on the cameras might help with another drawback highlighted by several of the team members: it was much easier to "zone out" and lose focus while working remotely. Or rather, as one of them put it: one is equally prone to losing focus "in the moment, but [working remotely] it is easier to find something else to do, and then you're mentally absent for longer". Two of the team members also noted that mobbing from home was slightly worse from an ergonomic point of view: on the one hand, you work from your own personal setup, but on the other hand you're stationary for longer periods at a time.

While reports of application of mob programming practice vary in related work, the studied team stands out in two respects: their level of informality with regards to roles, and the amount of time spent in session (full days without organized breaks, every day). Buchan and Pearl (2018) provide some quantitative data on the amount of time spent mob-

[2]The Pomodoro Technique is a time management method using a timer to break down work into intervals interspersed with short breaks, for the purpose of increasing focus and productivity.

bing in the form of online survey responses from active practitioners. Of 82 respondents, 51% reported that mobbing was their "predominant mode" of development. In other words, about half of the respondents do most of their work outside of the mob. That being said, the mob sessions in the studied team match the typical size of 4-8 participants reported by Buchan and Pearl (2018).

4.3. Interpersonal Context

Many of the interviewee remarks touched on **Social** aspects. Several recognized that mobbing had led to increased conflict within the team, particularly early on. The predominant view was that these conflicts weren't necessarily a bad thing — even though "they can be rather stressful" — while others were made to feel intensely uncomfortable by them. Interestingly, the interviewees provide examples of conflict which can be categorized both as task oriented (e.g. different views on appropriate design) and relationship oriented (e.g. the driver not listening to the navigators). Just as the interviewed mob participants themselves point out, a certain level of task oriented conflict can be beneficial in leading to improved judgment (Schwenk, 1990) and better decision-making (Schulz-Hardt et al., 2002), but only to the extent that such conflicts do not become too intense (Carnevale and Probst, 1998), implying the importance of managing the conflict level. This is emphasized by multiple studies showing that team member satisfaction as well as performance correlates negatively with conflict of either type — a finding that can be explained by high levels of conflict imposing a cognitive load on the affected individuals — and that the negative correlation between relationship conflict and satisfaction is particularly strong (De Dreu and Weingart, 2003).

It is clear from the interview responses that sensitivity to both task and relationship conflicts varies from individual to individual, and that these conflicts were a strong reason particularly in one of the team members' decision to reduce their participation in the mob. This raises the question of whether mob programming may inadvertently select for individuals less sensitive to task and/or relationship conflicts, unless these conflicts are properly managed. One coping strategy employed in the team to manage the level of conflict was to hold daily retrospectives, which served as pressure valves for the team, whereas "before we were able to build stress and tension for a week [in between retrospectives]". One of the team members called for "a more concrete way of handling conflicts [...] a person or a framework or something like that". It is worth noting that concrete advice on adopting an appropriate mindset and developing conducive habits is available in literature (Pearl, 2018), and even though a more formal adoption of such recommendations was called for in the interviews, the team had so far taken an improvised approach in this regard.

There was a clear split in the team on how they viewed the fact that the entire team did not participate in the mob. While acknowledging that "the team is a mob with a number of satellites" some team members didn't see a major issue with this, e.g. finding that "we have a better team spirit because now we're collaborating", others found that the team spirit had lessened as a consequence of mobbing. Related to this, one interviewee noted that a few team members joined after mobbing was introduced and were therefore never included in the decision to adopt that way of working, possibly affecting their readiness to buy into the practice. Even so, with some notable exceptions the majority of comments were positive: there were certainly issues, but in the end "people are quite kind" and "[it is] like we are a group of friends programming together". Several expressed joy at truly collaborating as a team, particularly early on. As one interviewee put it, "Previously we were a group of people, as is so often the case in these places. We call it a team, but it is really a financial construction, a number of employees who share a room."

A number of interviewee remarks concerned the **Organization**, both internal and external to the team. It was clear that the larger organization (comprising many dozens of teams) and its processes were not in any way adapted to the team practicing mob programming. Several of the team members described how expectations from "feature owners" on the team to work on multiple features in parallel clashed with the mob's singular focus on one task at a time. At the time of the interview, after nearly a full year of mob programming, the team still persevered and had learned to deal with the conflicts of interest, but the underlying incompatibilities between the processes of the team and the larger organization were still unresolved. With regards to internal organization, the importance of the rotating "supporting pair" in order to preserve responsiveness to incidents and customer requests while protecting the mob from distractions was repeatedly emphasized. The team had not tried ac-

tively controlling the constellation of members participating in the mob, e.g. to fit a certain task or to optimize learning, "but could surely have done so", which by some was seen as an option for addressing the fact that "it seems hard to ask questions when you're not keeping up, [because] you slow down a lot of people if you begin asking too much or confessing that you can't keep up". Nearly all of the interviewees touched upon joining and/or leaving an ongoing mob session. Some perceived it as "a difficult threshold" to cross to interrupt the mob by joining mid-session and forcing them to explain what they're in the middle of, whereas others reported that joining mid-session "hasn't been a problem; as long as you feel like you want to join". Conversely, several members felt that "once inside [you] are stuck until the mob stops", and while the team has at times tried to make a habit of more thoroughly introducing newcomers, one member reflected that "pretty often we forget".

In related work, the reports available on social and organizational aspects are largely anecdotal, with a lack of quantitative data.

4.4. Individual Factors

From the interviews it soon became clear that the degree of **Participation** varied between members of the team, and that while participation was encouraged within the group it was not mandatory. With just over half the team reporting that they participated to a high degree, a few rarely or never participating at the time of the interviews and two team members dedicated to troubleshooting and customer support (a rotating responsibility), mobbing sessions would usually include one third to just over half the team (i.e. 3-5 people). Those who participated to a lesser degree cited the shared (and therefore not to their liking) development environment, getting "stuck" with individual tasks and difficulties coping with the mob programming situation as such as reasons. It is important to note that even those who participated to a lesser extent at the time of the interviews had at other times been very active, e.g. starting out enthusiastic and participatory to then successively withdraw from mobbing. One consequence of these challenges was that the mob might lack needed competence: "the idea of a mob is that you should have at least someone [with insight into the particular challenge at hand]". Interestingly, some members felt that this caused "tension from the people inside the mob" while others claimed there were no tensions at all. We also note that in the studied team there is no sign of correlation with the developers' experience: both the most and the least active mob participants include a mix of the most and the least experienced team members. Similarly, we cannot draw any conclusions with regards to age or gender.

Related to participation, it was clear from the interviews that the team members had contrasting views on the importance of mob participation and the tensions and conflicts that might arise in the mob, and a number of them speculated on the role of **Personal Attitude** and personality. With regards to conflicts it was stated that "it depends on whether people are able to share opinions [and accept those of others]" and "everybody can't have their own way, you have to compromise", whereas one reflected that "I might be a bit conflict averse, and there have been conflicts in the mob that made me uneasy". Participation was largely found to be dependent on one's personal involvement in the way of working: "you should care about what the mob is doing", but some "prefer to finish [their individual] tasks first before joining the mob". One noted that for senior team members it came down to a question of prioritization between supporting the mob by sharing their experience or progressing faster on their own.

Regarding **Sustainability**, the two largest subthemes were ergonomics and exhaustion. On the topic of ergonomics, none of the team members experienced any significant issues — rather the opposite: "the best seated position is 'next'", and in the mob one is constantly changing positions. One remarked that having to sit down as a driver caused back pains, but "in theory that's easily solved" with a better desk. When it came to exhaustion, the team members had very different experiences: from "being completely exhausted" because "you have to stay focused 100% of the time" to it being "fairly relaxing" as you can "zone out a bit in between rotations". Some felt that it was initially exhausting, but improved as they learned to adjust, while others felt that they initially didn't feel any tiredness at all because mob programming was so fun and exciting, but that it set in over time. Still others reported feeling more tired after a day of mobbing than they would otherwise, but that it didn't bother them.

As noted in Section 3, individual factors such as personality traits are salient in related work. While there are no reported studies of e.g. correlation between personality traits and mobbing experiences and/or effectiveness, Buchan and Pearl (2018) re-

port survey responses on perceived impact of effective working in a mob, finding that 57% of respondents believe "Openness to new experience" to be the most important trait. Interestingly, none of their respondents picked "Degree of extroversion" as the most impactful trait, despite the attention given to this in other sources (see Section 5.3).

5. Analysis and Discussion

This section presents the analysis of the interview results, connecting back to the Areas of Interest in Table 3.

5.1. AoI_1: Productivity

The results with regards to overall productivity are ambiguous (with large differences between types of tasks, as discussed in Section 5.5). Several interviewees believe that mob programming does increase productivity, at least in the long term, but quantitative evidence of this fact is lacking. Others emphasize lead time optimization, and present credible arguments, even though quantitative evidence is still lacking. While not directly related to productivity, increased quality was similarly argued to be a consequence — also based on "a feeling". That being said, we argue that if the actual increase in productivity had been anywhere near the amazing figures sometimes reported (Zuill and Meadows, 2014), this would have been abundantly clear to everybody in the team. In other words, there is reason to believe that any productivity gains in the studied team were modest. One factor in this may be that the larger organization surrounding the team was clearly not adapted to derive any benefit from the features of mob programming, but rather struggled against them.

5.2. AoI_2: Learning

Accelerated learning was discussed by six sources (more than any other reported benefit, see Table 2), and it was largely supported by the interviewees. Even so, the view that this learning may be superficial in nature is interesting: being "spoon-fed" might prevent a deeper level of understanding. This hints at mobbing resulting in a more passive learning style than independently seeking information and learning by trial and error. This is relevant, as research suggests that differences in learning style affect cognitive outcomes (Michel et al., 2009). On the other hand, the mob situation is reminiscent of cooperative learning, with an emphasis on "learner-to-learner interactions" (Haidet et al., 2004); such cooperative learning is also claimed to be "the key strategy" for achieving active learning (Johnson and Johnson, 2008). It is also important to note that one's ability to learn within the mob depends both on one's own level of expertise relative to that of the mob, on the extent to which the team makes an effort to introduce newcomers, and on one's own attitude and behavior. Ultimately, the effects of mob programming on learning is an open question, with outcomes presumably dependent on multiple contextual variables. We consider the current lack of quantitative data on learning outcomes in mob programming to be a challenge in addressing this question.

5.3. AoI_3: Discomfort and/or Exhaustion

A striking feature of the interview results is the difference in perspective between the team members — there is clearly a factor of personal preference at play, with regards to how one experiences and feels about interaction with colleagues at the level of intensity that mob programming implies. This reflects the experiences of Schartman (2014) posted in a blog entry: "Being a fairly introverted person myself, the constant interaction is more of an energy drain than a lot of my previous work at the company, despite possibly being more fun. By the end of the work day I often find myself anxious to get home and recharge". Clearly this is not a universal experience, as evident from the conflicting interviewee statements. An online survey conducted by Buchan and Pearl (2018) found that the degree of extroversion was not considered by the respondents to be important for one's effectiveness in a mob. One interpretation of this is that unless personally affected, the importance of this factor is not evident: unless you feel it, you don't see it. Another interpretation, suggested by Schartman's account, is that one may still be *effective* in the mob even while being personally drained by it (possibly due to an introverted nature, which would be significant as previous work has found skilled software engineers in particular to tend towards introversion (Turley and Bieman, 1995)).

Although it is presently unclear how common the phenomenon is, what is clear is that some struggle with mob programming on a personal level. This is in line with findings in previous work related to exploratory testing — a practice we argue is in some ways analogous, as discussed in Section 1 —

where some organizations would assign testers to either exploratory testing or more traditional types of testing depending on their personality and preferences (Mårtensson et al., 2019). The interview results suggest that a similar approach to mob programming would be worth exploring: acknowledging that it may not be for everybody and creating flexibility in the organization to participate in mob programming to a greater or lesser degree, or even not at all, according to preference.

With regards to physical discomfort, the interview results largely conform with previous reports in literature. Consequently, we do not view this as a prioritized topic of research specific to mob programming — rather, as with rotations and breaks (see Section 5.6) — it is for practitioners to be aware of and to find solutions that work for them as individuals.

5.4. AoI_4: Social Dynamics

The team members' perceptions of how mob programming affected relationships within the team diverged markedly. While there was consensus that conflicts had increased, whether this was stressful or even negative at all was not agreed upon. Indeed, the larger, underlying conflict appears to have come from incompatible views on mob programming itself: in the studied team, it is clear that this drove a wedge deep into the team, creating "a mob with a number of satellites".

Reminiscent of previous findings from exploratory testing (Mårtensson et al., 2017), several team members felt joy at being given the opportunity to truly work together as a team and collaborate towards a shared goal, as opposed to being a mere "group of people" sharing a room and a backlog. This joy alone is a strong argument for any organization seeking to retain talent, with the caveat that while it is clearly felt by many, it is not felt by everyone. This was made evident e.g. by the contrasting views on its impact on team spirit. While some found that mob programming enhanced team spirit, others found it to be detrimental. One interpretation of this is that the interviewees had different notions of the boundaries of the "team". Within the inner circle of active mob participants, the team spirit undoubtedly increased. In the wider sense, though — among the "number of employees sharing a room" — several interviewees found that the team spirit had lessened. From the inside looking out, this may not seem like a pressing concern, but for the larger organization with a responsibility with regards to every employee's well-being it cannot be neglected. This challenge once again points to the importance of flexibility in allowing employees to participate in mob programming to the extent that they are comfortable with, without adverse side effects on team cohesion or other social considerations by — as Cain (2012) puts it — avoiding "the madness for constant group work".

It is worth considering whether these conflicts were truly *caused* by mob programming, however, or rather forced to the surface by it. A possible interpretation is that mob programming forged an actual team from a subset of the loose collection of "employees who share a room", its only failing being that some individuals were left behind as it were, thereby highlighting already present fracture lines within the team.

5.5. AoI_5: Task Suitability and Mob Composition

Just as the interview results point to organizational flexibility in adopting mob programming, they also suggest that mob programming is best applied to certain tasks while others are done individually, as discussed by several sources in the literature review (see Section 3). This arguably makes intuitive sense: it is difficult to discern any potential benefit from having an entire team watch a single person perform a simplistic task where there is nothing to discuss. Rather, one of the main benefits of a mob appears to be the ability to accrue enough competence to cover the entire problem domain, thereby cutting lead times caused by e.g. seeking advice from domain experts. One consequence of this is that additional participants, who do not add any new coverage of the problem domain at hand are in a very narrow sense superfluous. This suggests the potential to actively tailor the mob to fit each individual task and achieve competence coverage while avoiding "redundancy". To exemplify, it may be efficient for two members with a high degree of overlap in competence to work on separate tasks (e.g. in two different mobs), yet sometimes work together to maintain an open exchange of ideas — recognizing that short-term efficiency is not the only goal, and that learning potential may also be considered when considering mob composition. Granted, such perfect mapping of individual competences to each problem domain is rarely feasible, but may nevertheless serve as a model for composing a given mob.

One concern would be that — as suggested by the interviewees — the intensity and frequency of

conflicts peak in the early days of mobbing (see Section 5.4). This means that constantly forming and reforming mobs for each task could lead to a perpetual state of heightened tension, with the caveat that it is not clear to which extent those conflicts are actually caused by mobbing.

Finally, the observation that mob programming evens out the performance of the group by improving "the below-average and [pulling] down the above-average" raises an intriguing — and controversial — question. On its face it seems like a naturally good thing to elevate the floor, as it were, but presumably that depends on the relative difference in performance between the team members. It has long been noted (if not uncontested) that across professions, a minority of individuals tend to produce a disproportionate share of the value (Augustine, 1983), a phenomenon that if anything appears to be even more pronounced in software engineering (Curtis, 1981; Mills, 1988; Turley and Bieman, 1995). While answering it is outside of the scope of this study, this finding raises an interesting question: to the extent that the value of individual developers truly does vary as reported in some sources, is it in the organization's interest to even it out by including "exceptional" engineers in mobs? As one of the interviewees put it, this boils down to a question of prioritization between one's individual productivity and supporting the team.

5.6. AoI_6: Role Rotations and Breaks

Unsurprisingly, our study shows that people are different and have different preferences, both with regards to roles within the team and when and how to take breaks. Given the divergent views in literature and the interview results, we argue that these are factors that are easily experimented with within the individual team, and see no evidence that they are so crucial that they will make or break mob programming. Therefore we do not call on researchers to find a demonstrably superior practice, but on practitioners to be aware that there are many alternatives, and actively seek the optimum that works best for them as a group of individuals.

6. Research Roadmap Proposal

While we conclude that some of the original questions derived from related work are best answered by the individual practitioners in each respective case (see Section 5.6), a number of non-trivial questions with potentially general answers present themselves in the analysis in Section 5. This opens up multiple promising avenues for further research, leading us to propose research questions to address related to the organizational enablement of mob programming, its social dynamics, collecting quantitative data on its efficacy and how it may be effectively applied in remote work. We do not find that these areas are interdependent — and so do not form a step-wise path — but may be pursued independently and in parallel.

6.1. Organizational Enablement

A crucial concern is how the larger organization may enable, rather than counter-act mob programming practice. Questions to address within this area include:

- How may the organization enable members to participate to a greater or lesser degree, according to individual preference, without social stigma or other adverse side-effects?
- How may the organization surrounding the mob programming team benefit from, rather than clash with, its alleged lead time optimization and singular focus on one task at a time?
- How may the mob constellation be effectively and efficiently tailored to optimize current tasks, circumstances and goals, without suffering adverse effects?

6.2. Social Dynamics

Mob programming brings the social dynamics of the team to the fore. A deeper understanding of them is important both for the efficacy of the practice and the well-being of the practitioners:

- To what extent does mob programming cause new conflicts within a team, and to what extent does it surface latent conflicts?
- How may conflicts between team members be effectively handled in a mob programming paradigm?
- How do the personality traits of team members correlate with their experience of mobbing, particularly with regards to exhaustion and sensitivity to conflict, and how may mobbing be perceived as less exhausting to them?

6.3. Quantification

While significant claims have been made regarding improved productivity, learning, lead time optimization and quality, quantitative data would be helpful in supporting these claims:

- What quantitative data may be obtained on the short-term and long-term consequences of mob programming on productivity, learning, lead time optimization and/or quality?

6.4. Remote Work

While not included in the original planning, the covid-19 pandemic struck while this study was carried out, highlighting the question of mob programming while working remotely:

- How may the focus and sense of team presence of a physical mob programming session be preserved in a remote work context?
- How may remote mob programming be supported by improved tooling, e.g. making it easier to follow along with scrolling text in an editor over a high latency connection?

7. Threats to Validity and Reliability

This section discusses threats to validity and reliability.

7.1. Construct Validity

A potential threat to the construct validity of the study is that the reported experiences — e.g. of conflict within the team, or friction in interactions with external stakeholders — are actually not related to the practice of mob programming, but would have manifested themselves regardless. This possibility cannot be ruled out within the confines of the study itself, and we are therefore cautious about drawing definitive conclusions. Rather, we identify potential factors, analyze them, discuss possible interpretations and pose questions for further research.

7.2. Internal Validity

We find the three most salient threats to internal validity to be *ambiguity of causal direction*, *selection* and *mortality*. With regards to the first of these, we are careful not to make unsupported claims regarding the causality of observed phenomena. As for selection, to avoid any bias we set out to interview all of the members of the studied team, regardless of their involvement in or attitude towards the practice. One of the team members ultimately abstained, however — a threat of mortality — due to health reasons. Upon careful examination of the circumstances we see no reason to believe that this abstention was related to their view of mob programming or that their experiences would have deviated radically from the rest of the team members.

Related to *selection*, the fact that not all of the interviewees participated equally in the mob can also be viewed as a threat. Even if some did not participate in the mob at the time of the interviewees, all had participated enough to be able to provide a first-hand account of their experiences, both from within the mob and of the mob interacting with the larger organization. We would further argue that it is important to include the voices of those individuals who, based on their personal experiences, have chosen to withdraw from the mob in order to capture a full-spectrum view of the phenomenon.

7.3. External Validity

The interviewees in this study all come from a single development team, posing a threat to external validity: every team is different, with different internal dynamics, and there may be contextual factors affecting the experience of mob programming in this particular team. In response to this, we point to two facts. First, mob programming is yet an emerging field of research, and access to comparable teams with long-running experience is limited. Second, the interviewee statements largely conform to reports in related work, suggesting that their experiences are not unique. That being said, as noted above, we seek to err on the side of caution when drawing conclusions. To exemplify, it is important to note that the forming of a team within the team (see Section 4.3) would likely not be a problem in *every* team, but warrants caution and further investigation.

7.4. Reliability

To guard against erroneous conclusions caused by incorrect application of the study methodology, the thematic analysis of the interview data was conducted separately by the two researchers. The results were then compared, any discrepancies discussed and their conclusions ultimately merged. To ensure the correctness of the interview data itself, all transcripts were read back verbatim to the interviewees.

8. Conclusion

This paper addresses the question of *When and how is mob programming an effective and efficient method of software development?* This is a multi-faceted question: effectiveness has multiple possible interpretations, from team member well-being to team productivity, and regardless of whether is effective in any given sense it may or may not be an efficient use of available resources. To address the question, the state of published literature was surveyed (see Section 3), leading to the identification of six Areas of Interest for further investigation: productivity, learning, discomfort and/or exhaustion, social dynamics, task suitability and mob composition, and role rotations and breaks. These Areas of Interest informed subsequent interviews (see Section 4) with the members of a development team having nine months' experience from daily mob programming.

Thematic coding analysis of the interview results revealed that perceptions of mob programming vary greatly between members of the team. The resulting thematic map was viewed through the lens of each of the six Areas of Interest (AoI_{1-6}) and possible interpretations considered for each area (see Section 5). In summary, we find that different types of tasks clearly differ in suitability for mob programming: interview results indicate that mob programming has positive effects on lead time optimization and productivity when applied to complex tasks covering multiple domains, whereas effects on productivity more generally are ambiguous and likely to be modest. Similarly, implications of mob programming on learning are ambiguous: though frequently raised as a benefit in literature, it is possible that it may lead to a more passive learning style. Individuals react differently to mob programming, however, not least with regards to exhaustion and to the social dynamics within the mob. This implies the need for organizations to be flexible in their adoption of the practice, allowing developers ot participate depending on preference and ability. Unless properly managed, the these individual differences may give rise to tension and the forming of teams within the team.

In addition to these results, we present a roadmap to encourage and facilitate further work in what we regard as an emergent but promising field, which has so far only scratched the surface of the practice. Many more questions remain to be answered, and we present a list of 10 salient questions to address, categorized as pertaining to one of four categories: organizational enablement, social dynamics, quantification and remote work (see Section 6).

In this way, we strive to support not only industry practitioners by offering guidance in their quest for efficacy and efficiency in their development practices, but also the research community by presenting a structured set of relevant questions to investigate.

Acknowledgment

The authors would like to thank all the participating engineers for their insights, patience and willingness to share their experiences and data with us.

References

Augustine, N. R., 1983. Augustine's Laws and major system development programs. American Institute of Aeronautics and Astronautics.

Aune, O. K., Echtermeyer, C., Sørensen, E., 2018. Mob programming: A qualitative study from the perspective of a development team. Research Gate.

Bach, J., 2003. Exploratory testing explained.

Balijepally, V., Chaudhry, S., Nerur, S. P., 2017. Mob programming — a promising innovation in the agile toolkit. In: America's Conference on Information Systems (AMCIS).

Boekhout, K., 2016. Mob programming: Find fun faster. In: International Conference on Agile Software Development. Springer, pp. 185–192.

Buchan, J., Pearl, M., 2018. Leveraging the mob mentality: an experience report on mob programming. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering. pp. 199–204.

Cain, S., 2012. Quiet (TED Talk). `https://www.ted.com/talks/susan_cain_the_power_of_introverts/transcript`, [Online; accessed 17-June-2020].

Carnevale, P. J., Probst, T. M., 1998. Social values and social conflict in creative problem solving and categorization. Journal of personality and social psychology 74 (5), 1300.

Curtis, B., 1981. Substantiating programmer variability. Proceedings of the IEEE 69 (7), 846–846.

De Dreu, C. K., Weingart, L. R., 2003. Task versus relationship conflict, team performance, and team member satisfaction: a meta-analysis. Journal of applied Psychology 88 (4), 741.

Dybå, T., Arisholm, E., Sjøberg, D. I., Hannay, J. E., Shull, F., 2007. Are two heads better than one? on the effectiveness of pair programming. IEEE software 24 (6), 12–15.

Falco, L., 2014. Llewellyn's strong-style pairing. `http://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html`, [Online; accessed 11-March-2020].

Gregory, J., Crispin, L., 2014. More agile testing: learning journeys for the whole team. Addison-Wesley Professional.

Haidet, P., Morgan, R. O., O'malley, K., Moran, B. J., Richards, B. F., 2004. A controlled trial of active versus passive learning strategies in a large group setting. Advances in Health Sciences Education 9 (1), 15–27.

Hendrickson, E., 2013. Explore it!: Reduce Risk and Increase Confidence With Exploratory Testing. Pragmatic Bookshelf.

Hohman, M. M., Slocum, A. C., 2001. Mob programming and the transition to xp. In: Proceedings of the First XP Universe Conference.

Hove, S. E., Anda, B., 2005. Experiences from conducting semi-structured interviews in empirical software engineering research. In: 11th IEEE International Software Metrics Symposium (METRICS'05). IEEE, pp. 10–pp.

Johnson, R. T., Johnson, D. W., 2008. Active learning: Cooperation in the classroom. The annual report of educational psychology in Japan 47, 29–30.

Kattan, H. M., 2018. Theory of altruism on software development practices patterns. In: Proceedings of the 19th International Conference on Agile Software Development: Companion. pp. 1–4.

Kattan, H. M., Goldman, A., 2017. Software development practices patterns. In: International Conference on Agile Software Development. Springer, pp. 298–303.

Kattan, H. M., Oliveira, F., Goldman, A., Yoder, J. W., 2017. Mob programming: the state of the art and three case studies of open source software. In: Brazilian Workshop on Agile Methods. Springer, pp. 146–160.

Kattan, H. M., Soares, F., Goldman, A., Deboni, E., Guerra, E., 2018. Swarm or pair? strengths and weaknesses of pair programming and mob programming. In: Proceedings of the 19th International Conference on Agile Software Development: Companion. pp. 1–4.

Kitchenham, B., 2004. Procedures for performing systematic reviews. Keele, UK, Keele University 33 (2004), 1–26.

Lilienthal, C., 2017. From pair programming to mob programming to mob architecting. In: International Conference on Software Quality. Springer, pp. 3–12.

Mårtensson, T., Martini, A., Ståhl, D., Bosch, J., 2019. Excellence in exploratory testing: Success factors in large-scale industry projects. In: International Conference on Product-Focused Software Process Improvement. Springer, pp. 299–314.

Mårtensson, T., Ståhl, D., Bosch, J., 2017. Exploratory testing of large-scale systems - testing in the continuous integration and delivery pipeline. In: International Conference on Product-Focused Software Process Improvement. Springer, pp. 368–384.

Michel, N., Cater III, J. J., Varela, O., 2009. Active versus passive teaching styles: An empirical study of student learning outcomes. Human resource development quarterly 20 (4), 397–418.

Mills, H. D., 1988. Software productivity. Dorset House.

Pearl, M., 2018. Code with the Wisdom of the Crowd. Pragmatic Bookshelf.

Robson, C., McCartan, K., 2016. Real world research. John Wiley & Sons.

Rooksby, J., Hunt, J., Wang, X., 2014. The theory and practice of randori coding dojos. In: International Conference on Agile Software Development. Springer, pp. 251–259.

Sankaranarayanan, S., Wang, X., Dashti, C., An, M., Ngoh, C., Hilton, M., Sakr, M., Rosé, C., 2019. An intelligent-agent facilitated scaffold for fostering reflection in a team-based project course. In: International Conference on Artificial Intelligence in Education. Springer, pp. 252–256.

Schartman, M., 2014. My Experience With Mob Programming. `https://engineering.appfolio.com/appfolio-engineering/2014/03/17/my-experience-with-mob-programming`, [Online; accessed 17-June-2020].

Schulz-Hardt, S., Jochims, M., Frey, D., 2002. Productive conflict in group decision making: Genuine and contrived dissent as strategies to counteract biased information seeking. Organizational Behavior and Human Decision Processes 88 (2), 563–586.

Schwenk, C. R., 1990. Effects of devil's advocacy and dialectical inquiry on decision making: A meta-analysis. Organizational behavior and human decision processes 47 (1), 161–176.

Shiraishi, M., Washizaki, H., Fukazawa, Y., Yoder, J., 2019. Mob programming: A systematic literature review. In: 43rd Annual Computer Software and Applications Conference (COMPSAC). Vol. 2. IEEE, pp. 616–621.

Sun, W., Marakas, G., Aguirre-Urreta, M., 2015. The effectiveness of pair programming: Software professionals' perceptions. IEEE Software 33 (4), 72–79.

Turley, R. T., Bieman, J. M., 1995. Competencies of exceptional and nonexceptional software engineers. Journal of Systems and Software 28 (1), 19–38.

Wilson, A., 2015. Mob programming-what works, what doesn't. In: International Conference on Agile Software Development. Springer, pp. 319–325.

Zuill, W., Meadows, K., 2014. Mob programming: A whole team approach. In: Agile 2014 Conference, Orlando, Florida.

Zuill, W., Meadows, K., 2016. Mob programming: A whole team approach. Leanpub.

Daniel Ståhl is software subject matter expert at Ericsson AB and an associate professor of software engineering at Linköping University, Sweden. He has a background of thirteen years of large-scale software development in the industry. He received his PhD degree from the University of Gröningen, The Netherlands, in 2017 and his MSc degree from Linköping University, Sweden, in 2007.

Torvald Mårtensson is distinguished engineer, test manager and technical fellow at Saab AB. He has a background of fifteen years in the aeronautics industry and another eight years in the telecom industry. His work has primarily revolved around systems integration and system testing of large-scale software systems. He received his PhD degree from the University of Gröningen, The Netherlands, in 2019 and his MSc degree from Linköping University, Sweden, in 1997.