Flow barriers: What prevents software developers from experiencing flow in their work

Saima Ritonummi [1], Valtteri Siitonen [1], Markus Salo [1] and Henri Pirkkalainen [2]

[1] *Faculty of Information Technology, University of Jyvaskyla, P.O. Box 35, FI-40014 Jyvaskyla, FINLAND*
[2] *Unit of Information and Knowledge Management, Faculty of Management and Business, Tampere University, P.O. Box 527, 33101 Tampere, FINLAND*

Abstract

Software development requires high problem-solving skills and creativity, making it a profession with good opportunities to become immersed in a flow experience. The characteristics of flow experience are absorption, enjoyment, and intrinsic motivation toward the activity. This study aims to better understand the barriers that prevent software developers from experiencing flow at work. Previous research has mostly examined software developers' productivity, flow being one component of productive workdays. This study addresses the research gap by exploring the barriers to experiencing flow in software development. A qualitative questionnaire was used to gather data about flow experiences from 405 respondents. The most prominent flow barriers that emerged from these responses were *interruptions, too easy, boring, or repetitive tasks*, *lack of opportunities*, *insufficient requirements*, *timetables and deadlines*, and *problems with technology or software*. The results suggest that there are many more flow barriers in software development than what have been discussed in the context of productivity. These findings open up an interesting avenue for researching flow experiences in the software development context. The implication for practice is uncovering common flow barriers in software development, which can help both developers and managers identify these barriers, try to mitigate them, and facilitate more flow experiences at work.

Keywords

Software development, software developer, flow experience, interruptions, productivity

1. Introduction

This paper examines what prevents software developers from experiencing flow in their work. Flow experiences are associated with activities that require high skills and provide increasing challenge [1]. As software development requires high analytical problem-solving skills and creativity [2, 3], developers can be expected to experience flow in their work. On the other hand, software development is complex: It requires high levels of coordination and has many uncertainties, such as challenges in gathering and stabilizing requirements and mastering technologies and tools that constantly evolve [4]. Also, shortage of talent may cause unreasonable expectations for those who work in software development [3]. Therefore, many factors can either facilitate or prevent flow experiences in software development. Efficiently addressing these challenges presents organizations and managers with an opportunity to recruit and keep highly skilled, dedicated, and motivated employees [5].

The overarching research theme has been software developers' *productivity*. Previous studies have addressed software developers' flow as one component of productive workdays. Unsurprisingly, interruptions and task/context switching have been found to reduce developers' productivity, which leads to experiencing less flow during the workday [6, 7, 8, 9]. The few studies that have specifically

Proceedings of the 8th International Conference on Socio-Technical Perspective in IS development (STPIS'22), August 19-21, 2022, Reykjavík, Iceland
EMAIL: saima.e.ritonummi@jyu.fi (A. 1); valtteri.m.e.siitonen@jyu.fi (A. 2); markus.tsalo@jyu.fi (A. 3); henri.pirkkalainen@tuni.fi (A. 4)
ORCID: 0000-0001-6708-7639 (A. 1); 0000-0001-7610-4584 (A. 2); 0000-0001-5229-0300 (A. 3); 0000-0002-5389-7363 (A. 4)

addressed flow experiences in software development discuss flow in the context of developer experience (developers' user experience) [10], intention to code [3], engagement in open-source projects [11], and developers' emotions and progression at work (being stuck or in flow) [8]. However, the research on software developers' flow experiences is still very scarce. Although research has been conducted on software developers' perceptions of productivity and flow being an important component of productive their workdays, a more holistic understanding of software developers' flow experiences is needed. Moreover, Pratt et al. [3] have suggested further research on the perceived challenge–skill balance (an important condition for flow) in the software development context. This paper addresses the research gap by examining what prevents software developers from experiencing flow in their work more often and what kinds of flow barriers are present in software development work. Due to the socio-technical nature of software developers' work [12], we are interested in the human–technology interaction (HTI) perspective and whether developers' flow is obstructed by technology- or software-related barriers – in addition to organizational and social factors.

The study was conducted through a qualitative, critical incident technique (CIT)-inspired questionnaire that included open-ended and closed-ended questions about flow experiences and challenges in software development work. CIT is a useful method for collecting descriptions of "critical incidents" that have a significant effect (either positive or negative) on individuals and their behaviors and activities [13]. The primary data were collected from Prolific, as online panels have been found to produce high-quality data [14, 15]. The 405 respondents described their flow experiences and what they thought prevented them from experiencing flow in their work. Based on the findings, software developers experienced interruptions, a lack of challenge, problems with technology or software, and insufficient requirements and timetables related to the development projects as the most obstructing factors that prevented them from getting into flow at work more often.

The contributions of this study are twofold. First, the findings provide insight into the different barriers to experiencing flow in the software development context. Both new themes that apply specifically to flow experiences in the software development context and themes consistent with previous flow literature emerged from the data. Second, these findings can help advance the research and understanding of software developers' flow experiences, especially the characteristics of knowledge work and organizational life that contribute to obstructing flow in development work. The results also provide useful and practical implications in the software development context for ways to facilitate more flow experiences at work.

The rest of the article is structured as follows: Chapter 2 presents the theoretical background on flow experiences, technology-enabled flow, and software developers' flow. Chapter 3 describes the research method, data collection, and data analysis. Chapter 4 presents the results. Chapter 5 discusses the theoretical contributions, practical implications, limitations of the study, and future research agendas.

2. Theoretical Background

This section presents the research on flow experiences and technology-enabled flow as well as an overview of how flow experiences have been addressed in the software development context. This study discusses developers' flow experiences (i.e., optimal experiences), as defined by Csikszentmihalyi [1]. Therefore, other software development-related concepts with the word 'flow' (e.g., development flow, workflow, task flow, knowledge flow, data flow, or information flow) are not discussed here.

2.1. Flow experience

Introduced by Csikszentmihalyi [16], flow experience is described as the optimal experience, a state of deep concentration, and enjoyment. Flow activities are *autotelic*, that is, intrinsically rewarding; the experience is an end in itself and not done because of external rewards. One characteristic of flow is that flow activities usually require effort and provide an increasing challenge (action opportunities). Thus, flow activities may initially be reluctant to perform. However, when the action starts to provide feedback on one's skills, the activity becomes intrinsically rewarding. Therefore, these optimal experiences usually occur when the body or mind is stretched to the limits and something difficult or worthwhile is accomplished [1]. The nine dimensions of flow experience consist of the conditions for

flow and flow characteristics. The conditions for flow include perceived challenge–skill balance, clear proximal goals, and immediate feedback about the progress being made. The characteristics of a flow state include intense and focused concentration, merging of action and awareness, loss of reflective self-consciousness, a sense of control over one's actions, distortion of temporal experience (passing of time), and experience of the activity as intrinsically rewarding (autotelic experience) [17,18].

Entering flow requires a balance between the perceived action capacities (skills) and the perceived action opportunities (challenge). The capacities and opportunities in the person–environment equation are subjective: the challenge arises from the selectively attended information (i.e., perceived opportunities for action). In the traditional model of flow, the "flow channel" describes the perceived challenge–skill balance and how one's perceived skills correspond to the challenge that the activity provides: if the challenge is too low in relation to skills, it leads to boredom; if the perceived challenge is too high, it leads to anxiety. Consequently, flow occurs when there is the right amount of challenge and skills to respond to this challenge [18]. Flow can also be presented in a quadrant model in which a low-challenge, low-skill situation leads to apathy, a low-challenge, high-skill situation leads to boredom, a high-challenge, low-skill situation leads to anxiety, and a high-challenge, high-skill situation leads to flow [17]. A more refined version of this quadrant model is a map differentiating the challenge–skill combinations into eight segments, where the quality of the experience intensifies within a quadrant the more challenge an activity presents and the more skills it requires in relation to the person's average level. The eight segments are apathy, boredom, relaxation, control, worry, anxiety, arousal, and flow [18].

People experience flow differently. An autotelic personality is motivated in high-challenge, high-skill situations (flow) and least motivated in low-challenge, low-skill situations (apathy), whereas non-autotelics are least motivated in high-skill, high-challenge situations. Autotelics also experience less stress and strain in flow than outside of it, which is the opposite for non-autotelics [18]. Another interesting aspect of flow experience is the paradox of work [1], a phenomenon that has been addressed in many flow studies (e.g., [19]). The paradox of work describes the contradiction of how work is often disliked, despite being more absorbing than many leisure activities. Participants in flow studies have reported a desire to do something else while working, but research results suggested the opposite: Work provides higher-challenge, high-skill situations and experiences of efficacy (i.e., flow) than leisure [18].

Although the conditions and characteristics of flow are widely agreed upon, there has been discussion about the nature of flow experience and how to measure it [3]. Based on Csikszentmihalyi's conceptualization of flow, Abuhamdeh [17] argues that flow should be conceptualized as a highly enjoyable, optimal state of consciousness that should be distinguished from the conditions that elicit it. Nakamura and Csikszentmihalyi [18] point out that, as the flow state is, by definition, an autotelic experience (an end in itself), studying the *consequences* of flow is irrelevant, as the outcome is not what makes flow activities rewarding. What is also characteristic of flow experience is that individuals describing their flow experiences usually do not report sudden transitions between flow and non-flow states, which is important to note when researching flow. Therefore, setting boundaries for what is considered flow and what is not can be described as somewhat arbitrary. One solution to this dilemma has been to allow the participants of studies to decide for themselves whether the experience they had was flow or not and, if so, to ask follow-up questions about that experience [17].

2.1.1. Technology-enabled flow

An imperative goal in designing successful information systems is to place equal importance on both technical and social issues [5]. The enjoyment of an intrinsically motivated, goal-directed activity with an increasing challenge [20] makes information technology (IT) use a great source of flow experiences. The five characteristics of flow associated with flow experiences in the HTI context are pleasure, concentration, control, exploration, and challenge [21, 22]. Sharafi et al. [21] describe how an optimal combination of engagement modes with IT can help to facilitate flow. These engagement modes include enjoying/acceptance (the task provides an enjoyable challenge), efficiency/productivity (the challenge is possible to master), and ambition/curiosity (curiosity to find new challenges). The engagement mode involves the evaluation of the object, locus of control between the subject and the object (perceived challenge–skill balance), and intrinsic or extrinsic focus of motivation [21]. Aside from engagement

modes, different types of behavioral and motivational orientations can also explain how a person experiences flow in the IT use context. Similar to the description of an autotelic personality [18] and how curiosity as a trait of an autotelic personality can indicate intrinsic motivation towards the use of technology [3], Sharafi et al. [21] suggest that an 'autonomous orientation' is expected to experience high levels of flow in IT use.

Closely related concepts to experiencing flow in the HTI context include cognitive absorption, techno–work engagement, and techno-eustress. Cognitive absorption [23] is a state of deep involvement with technology/software, and it is derived from many theories, including the definition of flow. Tarafdar et al. [24] introduce a technostress trifecta, which includes techno-eustress—"positive and affirmative consequences relating to IS use and work tasks" (p. 15). The positive outcomes of techno-eustress may include improved performance, increased efficiency, and enhanced innovations at work. Techno-eustress may also induce IT use-related heightened flow, enjoyment, and immersion [24]. Techno–work engagement is defined as a "positive and fulfilling well-being state or experience that is characterized by vigor, dedication, and absorption with respect to the use of technology at work" [25, p. 2]. However, techno–work engagement is a relatively new construct, and thus the relationship between techno–work engagement and flow experiences is still unclear [25].

2.1.2. Software developers' flow

Flow has been widely addressed in the work context (e.g., [19, 26, 27, 28]) and in the knowledge work context (e.g., [29, 30]). Especially in some professional groups, such as IT professionals (whose work is often complex and requires a high degree of concentration), deep involvement in activities that require a specific skillset and provide enough challenge enables the assigning of an intrinsic reward to the work one does [31].

Empirical studies addressing flow experiences in the software development context have mostly focused on productivity, presenting flow as one component of productive workdays. In these productivity studies, flow is mentioned when it comes to suggesting that developers should not be interrupted when *in flow*, but they do not address software developers' flow experiences per se. Specifically, these studies have focused on how interruptions and task/context switches can be optimized so that software developers can have more productive workdays and experience more flow as a result [6, 7, 8, 9, 32, 33]. The few studies that do address software developers' flow examine developers' intention to code [3], their engagement in open-source projects [11], and flow in developer experience (software developers' user experience) [10].

The flow state is an important part of "high-momentum tasks", such as software development [33]. Interruptions and task/context switching cause a high level of cognitive cost (cognitive load) and subsequent low performance for software developers [32]. Switches and interruptions include everyday distractions, such as "getting sidetracked to other tasks; getting stuck or bored by complex or lengthy repetitive tasks; receiving priority change requests from the management team; or even something as simple as a question from a co-worker" [33, p. 1]. The problem-solving nature of the work makes developers especially vulnerable to interruptions and distractions during immersion, whether they are external or voluntary self-interruptions, as the capacity of human short-term memory is limited. Interestingly, developers have been found to perceive external interruptions to be more disruptive than voluntary ones because they cannot control these interruptions and their timing, although study results have suggested that self-interruptions are actually more disruptive than external ones. Moreover, developers describe context switching as requiring ramp-up time and a switch in mindset, which is more demanding on cognitive resources, making switching more disruptive than other interruptions. However, task switching is unavoidable, and sometimes it can even increase developers' productivity [32]. Abad et al. [32] argue that all interruptions are not as disruptive and that task switching can be considered a skill rather than an obstacle to work.

Nevertheless, research findings have suggested that developers *feel* productive when they complete many or large tasks without interruptions or task/context switches during the workday [6, 32, 33]. Developers have been found to perceive their days as productive when they complete tasks or goals, have the opportunity to get into flow without many context switches with no or few interruptions and distractions, have no meetings, have clear goals and/or requirements set, and can plan their workday

ahead [6]. It has been suggested that models for measuring productivity should capture developers' work more holistically and consider the individual differences in what is perceived as productive or not productive and that such models could be used to provide more tailored support to developers to get into and stay in the flow at work [7]. One such application is using biometric measures to follow developers' range of emotions and how their emotions correlate with the perceived progress during tasks [8, 9]. Graziotin et al. observe and theorize the development process from an affective perspective [2, 34, 35], finding that sustained flow is one consequence of developers' happiness and that broken flow is one consequence of unhappiness [35]. For developers, the greatest effects of both happiness and unhappiness influence their development productivity and quality (cognitive performance), which include creativity, flow, and process-related performance [35].

In empirical research on software development, flow experiences have been addressed in the context of perceived productivity and performance [6, 7, 8, 9, 32, 33, 34, 35], enjoyment and fun [3, 11], and developer experience [10]. It has been suggested that enjoyment [3], having fun [11], and user experience (UX) [10] may contribute to developers' flow. Also, emotional states can indicate developers' progress (whether stuck or in flow) [8]. As only some of the factors that prevent software developers from experiencing flow in their work are known, this article addresses the research gap in investigating the barriers to flow experiences in software development from a wider perspective: What are the other barriers to experiencing flow, aside from distractions, interruptions, and task/context switches? In the following section, we discuss the research method and address the research gap.

3. Method

We conducted a qualitative questionnaire with CIT-inspired open-ended questions. The CIT, which was introduced by Flanagan in [36], is a qualitative method for collecting observations of human behavior about "critical incidents," which make either a significant positive or negative contribution to individuals' activities [13]. We chose to conduct a CIT-style questionnaire because it is a useful method when the research focuses on individuals' behavior in their job/role in relation to other individuals or entities, especially in contexts that are episodic in nature [37]. Although the questionnaire included questions about an outstanding flow experience (a critical incident), the question addressing flow barriers in software development was not about a specific experience but about the barriers to experiencing flow in software development work in general. Therefore, this paper did not apply all the criteria of a "full CIT" [37] but instead used an adaptation of CIT.

3.1. Data collection

To answer the research question, we created an online questionnaire using LimeSurvey, through which the participants were asked to describe an outstanding flow experience in their work. The questionnaire included both open-ended and multiple-choice questions about the flow experience. The questionnaire also asked questions about other challenges at work not discussed in this paper. The purpose of the questionnaire was to determine the factors contributing to flow experiences and the factors preventing flow from occurring in software development. As suggested by Abuhamdeh [17], the survey was designed so that the participants could describe an experience they felt was flow, and the closed-ended follow-up questions aimed to gather more information about the incident. The open-ended questions asked the participants to describe the experience in detail: what exactly enabled the experience, how more flow experiences could be enabled, and what are the barriers to experiencing flow in the respondents' work. The multiple-choice questions were designed to collect supplemental information about the experience: how long the experience lasted, how long ago it occurred, how often the respondent experienced a similar flow at work, the positive effect it had, and statements related to the conditions and dimensions of flow. The questionnaire was created in Finnish and then carefully translated into English, including proofreading by a professional. Before publication, the questionnaire was pretested by three software developers to ensure the understandability of the questions.

Data collection was conducted in two phases: December 2021–February 2022 and March–April 2022. In the first phase (pilot study), we contacted the largest software development companies based on reported annual revenue in Finland and globally via email, targeting people working in software

development at these companies. In the second phase (primary data collection), we collected responses from the Prolific online panel, which has been found to produce high-quality data [14, 15]. Although MTurk is probably the most commonly used online panel for collecting questionnaire responses, Prolific participants have been found to be more naïve and diverse and produce higher quality data than MTurk participants [14]. We used the same questionnaire design in both phases, with no significant changes except for adding two attention check questions, as suggested by the Prolific guidelines. We also included the following screening criteria in Prolific: respondents who work in software-intensive industries[2], a minimum submission approval rate of 97%, and a minimum of 20 previous submissions on Prolific. We obtained 59 and 346 responses from the first and second phases, respectively (n=405).

The respondents' ages ranged from 18 to 73 years, with a weighted average of 31 years. Among the respondents, 76% were male, 23% were female, 1% were other, and 2% preferred not to disclose. Half of the respondents had a bachelor's degree, one-third had a master's degree, and the remaining had other degrees of education. The majority (79%) reported being employed full time. The demographics of the respondents are presented in Table 1.

Table 1
Demographics (n=405)

Age		**Gender**	
≤ 24	24%	Male	76%
25–34	47%	Female	23%
35–44	17%	Other	1%
≥ 45	9%	Not disclosed	2%
Education		**Employment status**	
Doctoral degree	1%	Employed full-time	79%
Master's degree	31%	Employed part-time	8%
Bachelor's degree	50%	Entrepreneur	4%
High school or equivalent	17%	Freelancer	4%
Less than a high school education	1%	Student	2%
		Other	2%

Most of the answers were from Finnish (15%), Portuguese (15%), British (9%), Polish (8%), Italian (7%), South African (6%), American (5%), Spanish (4%), Mexican (3%), and Greek (3%) respondents. Examples of the job titles reported by the participants included backend developer, frontend developer, full-stack developer, game developer, IT manager, mobile developer, programmer, software engineer, software developer, and technical lead.

3.2. Data analysis

The answers included in the analysis were screened in two rounds. First, we excluded all respondents who did not work in software development as software developers, software engineers, software testers, or closely related roles, such as scrum masters or technical leads. Second, the following exclusion criteria for flow experiences were determined from the outcome of the first round: 1) responses that described *workflow* rather than a flow experience, 2) students whose experiences were related to studying or school projects and not to work, 3) descriptions that were too incoherent to interpret, and 4) responses that did not describe a specific flow experience (below a 3.0 average to the flow statements [39]), as the goal was to examine flow barriers in those who have experienced flow in their work. Although Abuhamdeh [17] points out the arbitrariness of setting boundaries to what is counted as flow and what is not, the answers excluded with the last criterion were those from respondents who had either misunderstood the assignment or described *workflow* (thus having a lower than 3.0 average to the flow statements). The two rounds of screening resulted in 405 answers chosen for analysis.

[2] We screened for software-intensive industries [36], which in Prolific included Software, Information Services and Data Processing, and Video Games

After choosing the responses for the analysis, the coding process was conducted primarily by the first author. Coding involved reading through the data and establishing categories using open coding, labeling the responses, and describing the phenomena that arose from the data [40, 41, 42]. The preliminary categories were established by coding the first set of data (n=59) and refined when coding the second set of data (n=346). The preliminary categories were roughly grouped into internal and external factors. Later, the perceived challenge–skill balance-related barriers were added as their own category between internal and external factors. Some of the initial topics were challenge–skill balance, interruptions and distractions, development project-related barriers, technology-related barriers, and mental unease (e.g., stress and anxiety). These were further defined after coding all the answers and the categories were discussed among all authors. The content analysis process involved the data-driven development of the main categories and subcategories. The objective was to provide insights into the frequency and patterns of the factors affecting flow experiences in software development work [42].

4. Results

The findings about the flow barriers in software development work were categorized into internal factors, perceived challenge–skill balance, and external factors. As most of the responses included descriptions of multiple barriers, they were coded with multiple labels. The categories were more or less linked to one another, and some had a certain degree of overlap. For example, distractions, interruptions, and task/context switching overlapped to some degree. In terms of frequency, the most prominent flow barriers in software development were interruptions (66), too easy, boring, or repetitive tasks (48), lack of opportunities (38), insufficient requirements (28), timetables and deadlines (28), and problems with technology or software (27). Most of the respondents reported experiencing flow sometimes (43%) or often (30%), the rest either very often (12%), rarely (12%), or very rarely (2%). Interestingly, 10 respondents answered that there were no barriers to experiencing flow in their work.

4.1. Internal factors

There were 66 mentions of internal barriers to experiencing flow related to mental unease, cognitive performance, and health. Mental unease barriers included a lack of motivation, stress, worries, anxiety, procrastination, and self-doubt. Cognitive performance included concentration. Health-related barriers included sleep and rest, health conditions, exercise, and nutrition. These are presented in Table 2.

Table 2
Internal flow barriers (66)

MENTAL UNEASE (48)	Lack of motivation (22) Stress (8) Worries (7) Anxiety (4) Procrastination (4) Self-doubt (3)
COGNITIVE PERFORMANCE (9)	Concentration (9)
HEALTH (9)	Sleep and rest (5) Health conditions (2) Exercise (1) Nutrition (1)

As expected, mental unease prevented developers from experiencing flow in their work. Mental unease barriers include a ***lack of motivation***, ***stress*** (both work-related and from outside work), ***anxiety***, ***worries*** (e.g., overthinking and problems in personal life), ***procrastination***, and ***self-doubt*** (trust in one's own skills, fear of not being enough, or fear of being misjudged).

"I still work from home, and that has made my home environment a lot more toxic. I fail to get work done sometimes because I just procrastinate, and I struggle to relax and just procrastinate because I feel anxiety from being in my 'workplace.'"

Lack of motivation was mainly described as a lack of passion, commitment, or interest in the tasks or toward the industry in general (e.g., tasks that are not fun and feel like "work" and "projects that are for-profit and not for fun"). Motivation-related answers overlapped with perceived challenge–skill balance (i.e., the lack of motivation is caused by not having enough challenge).

"The overall lack of motivation toward the video game industry in general, which is growing progressively."

Under cognitive performance, ***concentration*** was described as a lack of focus, being easily distracted, and having a hard time getting absorbed into something due to concentration problems. The responses labeled in this category ranged from a general lack of focus (which could result from being tired, for example) to descriptions of challenges that can refer to neurological challenges with attention regulation. There was also one response about how the war in Ukraine has affected the respondent's ability to focus on work.

"Sometimes I can't focus on a single task as I have many small things to do."

"*I have concentration problems, so it's really hard for me to get absorbed in something."*

Health-related barriers included ***sleep and rest*** (lack or bad quality of sleep, tiredness), ***health conditions*** (chronic and non-chronic problems with health), lack of ***exercise***, and not ideal ***nutrition*** for supporting flow (one respondent mentioned drinking too much coffee).

"… I also think I have to be fully awake and alert so if I'm not feeling great (and I have a chronic health condition), then that can stop me from experiencing flow."

4.2. Perceived challenge–skill balance

The perceived challenge–skill balance is an essential condition for flow. The responses categorized under perceived challenge–skill balance factors were characterized by a mismatch of an appropriate level of challenge provided by the work. The perceived challenge–skill balance is between the internal and external factors because it is a sum of the external and internal factors and the constant balancing between challenge (external opportunities for action) and one's skills to pursue the challenge [43]. There were 161 mentions of perceived challenge–skill balance, categorized as not enough challenge and too much challenge (Table 3). The barriers under not enough challenge were too easy, boring, or repetitive tasks, lack of opportunities, and lack of creativity. The barriers under too much challenge were constant learning, too-hard tasks, knowledge gaps, and a lack of skills or experience.

Table 3
Perceived challenge–skill balance (161)

NOT ENOUGH CHALLENGE (99)	Too easy, boring, or repetitive tasks (48)
	Lack of opportunities (38)
	Lack of creativity (13)
TOO MUCH CHALLENGE (62)	Constant learning (22)
	Too hard tasks (21)
	Knowledge gaps (10)
	Lack of skills or experience (9)

The most prominent barrier under not enough challenge was ***too easy, boring, or repetitive tasks***, which included descriptions of tasks that did not present enough challenge and tasks that are boring, mundane, insignificant, repetitive, or monotonous. The tasks that did not present enough challenge and/or peak interest were experienced as decreasing the likelihood of reaching flow at work and, in some cases, causing a lack of motivation.

> *"Because in tech, you work with boring tasks a lot, and a new greenfield project comes rarely. Even if it comes often, most of the projects will be boring stuff, or you will get yourself into a maze more often than not, which will decrease the flow."*

> *"Some of the projects we get do not challenge us. They are mostly simple things that we can do with our eyes closed. Therefore, one loses the motivation to do one's best because there is no challenge."*

A lack of opportunities was described as not being given a chance to prove one's skills due to the type of tasks or projects assigned (most of the respondents who mentioned this were junior developers), not having enough interesting projects to work on, the staleness of work, and lack of autonomy, new challenges or learning experiences. ***A lack of creativity*** was characterized by task- or project-related restrictions in creativity, innovation, and exploration. Some of the responses about a lack of creativity overlapped with those about too easy, boring, or repetitive tasks.

> *"My work includes a lot of mundane tasks, such as little alterations and additions to existing programs, so that they meet the new needs of clients. This requires no creativity on my part. It is not challenging, and frankly, it's plain boring."*

> *"Most tasks are time-consuming but not creatively demanding, so the opportunity to flex my creative muscles is limited."*

Constant learning was the most frequently mentioned barrier in the too much challenge category. It included descriptions of steep learning curves, learning to use the software while working on the task (which was described as interrupting the immersion), and the many software, frameworks, and programming languages required in development work. Moreover, learning "*the ins and outs of a new software*" combined with time restraints was found to be taxing, thus not supporting getting into flow. Constant learning was followed by ***too hard tasks***, which was characterized by being presented with too hard, challenging, complex, or frustrating tasks (or, as one respondent indicated, "extremely unmanageable tasks").

> *"Some tasks just seem difficult to boil down to a simple checklist and using only a few tools/services. And it's this simplicity in which I seem to experience the best flow."*

The ***knowledge gaps*** barrier was characterized as the inability to perform a task or solve a problem due to gaps in knowledge (e.g., having to do research before solving or fixing a problem). ***Lack of skills or experience*** was described as a feeling of not having enough skills or experience for the tasks that one is expected to master in their role or in the field of software development. The difference between these two barriers is that knowledge gaps are more task-centric, whereas a lack of skills or experience is a role-related barrier (i.e., not being skilled enough for one's job).

4.3. External factors

The most mentioned flow barriers were related to external factors: 386 mentions of interruptions and distractions, project-related reasons, organizational factors, technology, and circumstances as obstructing flow (see Table 4). Interruptions and distractions were further subcategorized into interruptions, distractions, and task/context switching. Project-related barriers included insufficient requirements, timetables and deadlines, working with clients, workload, getting stuck, waiting, and

unexpected changes. Organizational barriers involved management, company policies, team dynamics, resources, and communication. Technology-related barriers were problems with technology or software, low code quality, technical dept, use of non-preferred technology or programming language, and inadequate or missing documentation. Circumstance barriers included three categories: flow needs specific circumstances, workspace, and flow only happens at a certain stage of the development process.

Table 4
External flow barriers (386)

PROJECTS (118)	Insufficient requirements (28)
	Timetables and deadlines (28)
	Working with clients (16)
	Workload (15)
	Getting stuck (11)
	Waiting (11)
	Unexpected changes (9)
INTERRUPTIONS AND DISTRACTIONS (109)	Interruptions (66)
	Distractions (23)
	Task/context switching (20)
ORGANIZATION (71)	Management (21)
	Company policies (18)
	Team dynamics (17)
	Resources (8)
	Communication (7)
TECHNOLOGY (71)	Problems with technology or software (27)
	Low code quality (14)
	Use of non-preferred technology or programming language (12)
	Technical dept (11)
	Inadequate or missing documentation (7)
CIRCUMSTANCES (17)	Flow needs specific circumstances (9)
	Workspace (5)
	Flow happens at a certain stage of the development process (3)

The most addressed project-related barriers to flow experiences were insufficient requirements and timetables and deadlines. ***Insufficient requirements*** was described as a lack of requirements and/or unclear requirements. This barrier was usually mentioned as a result of insufficient planning, which led to having to develop, for example, applications without a "complete picture" and clear goals.

> *"Having unclear requirements so I have to constantly bounce back and forth between my IDE (integrated development environment) and other communication tools. Also, having meetings that interrupt my flow and productivity."*

Timetables and deadlines were described as time pressure, too tight deadlines, constant hurry, short time for implementation, lack of time (including lack of time for discovery), and fear of not being able to finish the work within a given timeframe. There was one mention of the "absence of little stress on deadlines," which implies that not having a deadline at all is not ideal for experiencing flow in software development work.

> *"I fairly regularly face tasks and challenges that I am not prepared to complete, coupled with tight timeframes."*

Working with clients included descriptions of working on clients' requests that either stress the whole team or do not provide enough challenge, clients who are not open to ideas provided by the

development team, having too many clients to work with, and poor interaction with clients. ***Getting stuck*** was related to running into problems and blockers that prevent progression with the task and losing time solving problems that take longer to solve than expected. ***Waiting*** included waiting for replies on submitted tasks, delays, not being in sync with the team (i.e., not finishing one's share of work on time causing delays for others), too slow feedback loops, and waiting for help. ***Workload*** was characterized by not being able to affect the workload coming in (either too much or too little work), pressure, multiple simultaneous projects, no possibility of taking breaks, and being daunted by the amount of work.

> *"There are so many bugs that feeling energized every time you fix an issue is just not realistic because your brain tells you that there's always something else."*

Changes was described as last-minute changes in projects, changing prioritization, unforeseen circumstances, unpredictable new challenges, and things not going according to plan or not working as expected. All of these aforementioned project-related reasons were the most prominent external category for not being able to experience flow more often in software development work.

In the interruptions and distractions category, ***interruptions*** was the most prominent barrier. Interruption-related responses described either physical or virtual interactions with team members ("too many meetings" and "unnecessary meetings" were recurring themes in interruptions). Interruptions also overlapped with task/context switching, mostly with meetings and other interruptions disrupting development work. There were 66 mentions of interruptions in total, making it the number one flow barrier mentioned by the participants. Among these mentions, 23 were about general interruptions, 19 about meetings, 16 about other interaction-related interruptions in the physical environment (interacting with colleagues and team members), and 8 about interruptions in the virtual environment (instant messaging, email, notifications, phone calls). The following response sums up most of the respondents' feelings about virtual interruptions:

> *"The order of terror: slack, teams, calendar bookings, email."*

Distractions were divided into physical distractions and virtual distractions. The mentions were mainly about physical distractions in the workplace (e.g., open-plan office and noise). There were also some mentions of working from home (family members and pets) and one about virtual distractions (internet). The ***task/context switching*** barrier included many comments about not having sufficient time blocks for coding due to task/context switching. Switching between many projects and jumping between tasks were described as the reasons for the inability to focus intensely on one thing for extended periods. As expected, there was some overlap between distractions and interruptions, as both caused fragmented workdays.

> *"There is too much context switching between projects, making it hard to deeply focus on one thing. Also, the usual management overhead, albeit important, tampers productivity."*

> *"Fragmented workdays. Previously, as a full-time software developer, time was better allocated for app development. Now, all sorts of questions from other developers and meetings break up the day. For me, a flow state requires an extended, uninterrupted moment."*

The most mentioned flow barrier related to organizational aspects was ***management***. This barrier was characterized by a lack of good leadership, micromanagement, pressure from above, management overhead, constant changes in leadership, ineffective routines that disturb development, unwillingness or resistance to implementing new solutions, lack of appreciation and recognition, and management that wants to do things their way (not what would be smart development-wise).

> *"Everyday distractions that are caused mostly by lack of organization by my managers."*

The ***company policies*** barrier was related to bureaucracy, strict procedures, company-imposed guidelines to use certain technologies or software, too many development practices, non-flexible

working hours, and a business model that does not allow developers to spend their working hours focusing on one task (which is related to task/context switching).

> *"The lack of cooperation or understanding of how things work as well as excessive bureaucracy in a lot of processes that can be resolved more easily."*

Team dynamics included comments about lack of cooperation, engagement, or teamwork among team members, competition among team members, too-big expectations, a toxic work climate, difficulty in getting help, and other people (especially superiors) crushing ideas. ***Resources*** was also an important factor for the inability to experience flow. This included a lack of resources, support, or training and working alone. The ***communication*** barrier was related to a lack of communication and poor communication.

Technology-related barriers included ***problems with technology or software*** (different kinds of issues with technology or software not working as expected), ***low code quality*** ("bad" or untestable code, bugs, and glitches), the ***use of non-preferred technology or programming language*** (either not getting the opportunity to use the preferred technology or programming language due to the nature of projects or company policies or not having found the preferred "fun and intuitive" programming language), ***technical dept*** (working with legacy code and technologies), and ***inadequate or missing documentation***, which hinders progression.

> *"Legacy code, bugs that need urgent fixing, use of old cumbersome tech."*

The circumstance factors included descriptions of how ***flow needs specific circumstances*** in which a mix of settings needs to be aligned for flow to take place (technical requirements are met, having enough time, mood, etc.). In other words, "it has to tick all the boxes."

> *"Usually things are 'okay,' but it seems to be a rarity when everything 'comes together,' when you are able to develop a large subsystem where almost everything works correctly the first time, without many bugs, and you are able to push the feature out the door and get it tested and into the hands of clients very quickly."*

The circumstance barriers also included comments about the ***workspace***, whether it was too hot, dirty, or chaotic, or working from home settings that do not support getting into flow. The respondents who indicated that ***flow happens at a certain stage of the development process*** explained that what prevents them from experiencing more flow at work is that they experience it only at certain stages of the software development process. More specifically, flow can be either related to development stage-specific tasks or moving to different stages of the development cycle can result in losing the flow.

Finally, four responses were labeled miscellaneous. Five respondents could not think of any barriers, and 10 reported not having any barriers to flow. These 10 responses included the following descriptions: *"Nothing is preventing me from experiencing flow in my work,"* "*Nothing. It happens to me very often,"* and *"There's nothing stopping me from experiencing flow more often in my work. It's just my choice."*

5. Discussion

A remarkable number of software developers reported how working in the software industry and the characteristics of organizational life prevent them from experiencing flow more often. Internal reasons were the least significant factor in their inability to experience flow at work. There were more than five times more mentions of external barriers (386) than of internal barriers (66). Interruptions were the most prominent flow barrier in software developers' work. These descriptions entailed interruptions in physical and virtual environments, such as meetings, colleagues asking questions, phone calls, email, and instant messaging. The second and third most significant barriers were related to the perceived challenge–skill balance: too easy, boring, or repetitive tasks and lack of opportunities. These were followed by insufficient requirements and timetables and deadlines (project-related barriers) and problems with technology or software.

As the flow state can be controlled to some degree and not just left to chance, and reaching continuous flow requires either physical exertion or a highly disciplined mental activity [1], it is not surprising that interruptions can be especially harmful in software development work. Human performance relies heavily on the self-regulation of attention, and when it comes to focusing, human cognitive resources are very limited [34]. One of the essential reasons developers have been found to feel productive at work is when they can complete their workday without significant interruptions, distractions, or task/context switches [6]. This same reason also seems to be the most important category of barriers to developers' flow experiences at work. Interestingly, developers tend to perceive externally imposed interruptions as more disruptive than self-inflicted interruptions [6, 33]. Accordingly, only one respondent in this study described self-inflicted interruptions as being able to disrupt their flow. Task/context switching has also been discussed in software development research. Although research has been conducted on the harmful effects of task/context switching on software developers' work (e.g., [8, 32, 33]), it is also an unavoidable characteristic of software development work. At times, the trade-off between perceived individual productivity and the team's productivity can even increase the developers' overall productivity [6, 33]. Some respondents in this study acknowledged this tradeoff in their descriptions of certain factors being harmful to their individual flow but important for a project or the team, for example. Nevertheless, most developers felt that interruptions and distractions prevented them from getting into flow.

Abuhamdeh and Csikszentmihalyi [20] propose that challenge is important for experiencing enjoyment in intrinsically motivated, goal-directed activities. The vast variability in individual skills has been a frequently emerging theme in research on software development work [4]. Pratt et al [3] called for further research on the challenge–skill balance in the software development context. Therefore, it is not surprising that two barriers related to work not providing enough challenge were among the most prominent flow barriers identified in this study. Remarkably, mentions about not having enough challenge were greater than those about having too much challenge. On the contrary, tackling too high a challenge in relation to one's skills can lead to anxiety [18] and be ineffective for both individuals and organizations [44]. This was present in some of the responses, including descriptions of having anxiety because of too-hard tasks. Although learning is an inherent part of software development work, some respondents seemed to feel that constant learning and steep learning curves prevented them from experiencing flow. One of the reasons this was considered a flow barrier could be that these learning experiences might not result in the expected performance level, which could frustrate highly skilled developers. Palomäki et al. [45] suggest that flow can be linked to better-than-expected performance (moderated by task experience) rather than to the absolute performance level and that better-than-expected performance is associated with more flow, whereas worse-than-expected performance is associated with less flow. Thus, the reason why too much challenge prevents developers from experiencing flow could be that worse-than-expected performance makes them question their skills and self-efficacy. It can also be that working at the limits of one's skills all the time can cause drain, and even an inability to perform [44].

In the project-related barriers, insufficient requirements were characterized by a lack of planning or a clear picture of what was being done (i.e., lack of clear goals). Gathering and stabilizing requirements are challenging in software development [4]. However, having clear proximal goals is an essential condition for flow [18], especially in software development [3]. Software developers have reported having productive workdays when they have clear goals and/or requirements [6]. Regarding timetables and deadlines, developers felt that too-tight timetables prevented them from getting into flow. Earlier research has also suggested that developers prefer the opportunity to plan their workday ahead [6]. Changing prioritization, unexpected changes, and interruptions or distractions combined with tight timetables do not foster flow at work. However, some deadlines need to be set. Having no pressure to meet deadlines can also be a flow barrier if one does not have any external demands to be productive, as described by some of the respondents.

The results also show that technology-related reasons, which are mostly related to having different kinds of problems with hardware or software (e.g., buggy code editor), can prevent software developers from experiencing flow in their work. This finding supports earlier research on developers' UX that, in working with different tools, such as integrated development environments (IDEs), developers appreciate the efficiency, informativeness, intuitiveness, and flexibility of the tool used [10]. The importance of UX was also shown in the mentions about having to work with non-preferred technology

or programming languages. Other technical difficulties obstructing flow included technical dept, low code quality, and inadequate or missing documentation, of which at least low code quality and missing documentation have been identified as linked to developers' unhappiness at work [35].

5.1. Theoretical contributions

The theoretical contributions of this study are twofold. First, whereas some of the emergent themes from the data-driven categories were consistent with previous software development research on productivity (e.g., interruptions) and flow theory (especially conditions for flow), the novelty value of this paper comes from the new emerging themes of flow barriers in the software development context. The most important categories of flow barriers in this dataset of software developers were interruptions and distractions, not enough challenge, and project- and technology-related reasons. Among the categories, interruptions and distractions have been previously examined as related to flow experiences in developer productivity studies. Also, some of the technology-related flow barriers are linked to the findings of developer experience and developers' (un)happiness at work.

Second, whereas previous studies have addressed software developers' flow mostly from a productivity point of view [6, 7, 8, 9] or in the context of UX [10], this study investigated software developers' flow and its barriers through the flow lens. The study was designed to understand the characteristics of developers' flow and how they describe flow in their own words. Hence, the questions were not contextualized to productivity at work or to the HTI perspective specifically. The findings contribute to software development research by investigating developers' experiences at work, thereby increasing our understanding of and ways to enable optimal experiences in software developers' work.

5.2. Practical implications

In terms of practical implications, the most apparent challenge seems to be finding an ideal balance between perceived challenge and skill combination in which the developer can use their potential but is not faced with excessively difficult tasks or steep learning curves too often. Being either too close to boredom or anxiety in one's competence zone (where flow can happen) can result in being less effective at work [44]. In the case of not having enough challenge, developers reported that they would like to be given a chance to show their skills and to have more meaningful and interesting tasks and projects to work with. Although many developers are proactive in choosing projects and tasks that provide the right amount of challenge whenever possible, it is vital that organizations and management coordinate their expertise accordingly. Work can be made a greater source of flow by shaping the activity structures and environment so that they foster more flow and obstruct it less [18], for example, by trying to deal with too complex tasks and problems with more organizational support and resources. Also, a better understanding of flow could enable organizations to recruit more employees who demonstrate intrinsic motivation and curiosity, which is characteristic of autotelic personalities [3].

More concrete flow barriers are easier to identify and address. These include reducing interruptions, being more specific about gathering requirements, keeping timetables and deadlines manageable, and mitigating problems with technology or software whenever possible. One respondent mentioned the maker's schedule and the manager's schedule [46], referring to the different needs of software developers (makers) and managers for productive workdays. In this study, there were many mentions of a lack of extensive time blocks for development work resulting from different interruptions, mostly meetings and task/context switches. For makers who need uninterrupted time for focused problem solving, task/context switches and meetings breaking up the workday are very costly. By contrast, for managers, speculative meetings and interaction-related interruptions can instead be an opportunity. As interruptions were software developers' most prominent barrier to getting into flow at work, it is beneficial if both developers and management are willing to compromise and acknowledge that a certain number of meetings are required but that they come with a high cost of interruptions for the developers [46]. In supporting developers' technology-enabled flow, one area of improvement is development environments that provide useful affordances and foster flow experiences rather than obstruct them. Better-designed IDEs have been suggested as one way to enable more flow experiences and boost intrinsic motivation in the software development context [10, 47].

5.3. Limitations

The most obvious challenge related to studying flow is that it is a subjective experience, and it is impossible to measure a subjective experience directly [29]. Another limitation related to qualitative content analysis is that it is ineffective in testing causal relationships between variables [40]. Other possible challenges of a CIT-style questionnaire study include issues of reliability and validity, misinterpreting or misunderstanding the participants' stories, ambiguity associated with category labels and coding rules, and recall bias [13]. In addition, it is possible that not all flow barriers in the software developers' work emerged in this study. Therefore, more factors could be discovered if a more extensive dataset is used.

The limitations specific to collecting responses from online panels include the inability to verify respondents' identities, the chance of respondents not giving their full attention to the task, and "super users" who may distort results [48]. Although the respondents' identities could not be verified, we tried to mitigate the other shortcomings by using prescreening and elimination criteria to sort out possible super users and respondents who did not give their full attention to the tasks.

5.4. Future research agendas

An interesting avenue of research for examining possible facilitators or contributing factors in flow experiences is personal characteristics, such as autotelic personality and its role in experiencing flow in software development. Individuals who work in software development may have autotelic personalities and enjoy the high-skill, high-challenge environment [18] that the work provides. In addition, as flow experiences are associated with a highly skilled activity, how flow is related to learning new skills should be examined more [45]. Being a highly skilled professional and yet having to constantly learn new skills as a software developer could be a topic of interest in future flow studies.

Another theme for further research is collective or social flow [29, 44, 49]. To the best of our knowledge, collective flow has not yet been addressed in the software development context. It would be interesting to study how teamwork and meetings could be balanced with sufficiently extensive time blocks for coding so that a collective flow could be reached, instead of these features of organizational life being barriers to individual flow.

Moreover, the relationships between software developers' flow and closely related concepts, such as cognitive absorption [23], techno-eustress [24], and techno-work engagement [25], could be researched further. Most importantly, both the literature review and the findings of this study suggest that there is much more to know about software developers' flow, including its antecedents, characteristics, and barriers.

6. Acknowledgments

This research has been funded by the Foundation for Economic Education, Finland, and partially by the Academy of Finland (341359).

7. References

[1] M. Csikszentmihalyi, Flow: The Psychology of Optimal Experience, 1st ed., HarperCollins, New York, NY, 1990.

[2] D. Graziotin, X. Wang, P. Abrahamsson, Happy software developers solve problems better: Psychological measurements in empirical software engineering, PeerJ 2 (2014) 1–23. doi:10.7717/peerj.289

[3] J. A. Pratt, L. Chen, C. Cole The influence of goal clarity, curiosity, and enjoyment on intention to code, Behaviour & Information Technology 35 (2016) 1091–1101. doi:10.1080/0144929X.2016.1171399

[4] S. Kudaravalli, S. Faraj, S. Johnson, A configural approach to coordinating expertise in software development teams, MIS Quarterly 41 (2017) 43–64. doi:10.25300/MISQ/2017/41.1.03

[5] E. Mumford, A Socio-Technical Approach to Systems Design, Requirements Engineering 5 (2000) 125-133. doi:10.1007/PL00010345
[6] A. N. Meyer, T. Fritz, G. C. Murphy, T. Zimmermann, Software developers' perceptions of productivity, in: Proceedings of the 22nd ACM SIGSOFT Symposium on the Foundations of Software Engineering, SIGSOFT/FSE '14, ACM Press, New York, NY, 2014, pp. 19–29. doi:10.1145/2635868.2635892
[7] A. N. Meyer, E. L. Barton, G. C. Murphy, T. Zimmermann, T. Fritz, The Work Life of Developers: Activities, Switches and Perceived Productivity, in: IEEE Transactions on Software Engineering, 43 (2017) 1178–1193. doi:10.1109/TSE.2017.2656886
[8] S. Müller, T. Fritz, Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress, in: Proceedings of the 37th IEEE International Conference on Software Engineering, IEEE/ACM '15, 2015, pp. 688-699, doi:10.1109/ICSE.2015.334
[9] T. Fritz, S. Müller, Leveraging biometric data to boost software developer productivity, in: Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER '16, 2016, pp. 66-77. doi:10.1109/SANER.2016.107
[10] K. Kuusinen, H. Petrie, F. Fagerholm, T. Mikkonen, Flow, intrinsic motivation, and developer experience in software engineering, volume 251 of Lecture Notes in Business Information Processing, International Conference on Agile Software Development, Springer International Publishing, 2016. doi:10.1007/978-3-319-33515-5_9
[11] B. Luthiger, Fun and software development, in: Proceedings of the First International Conference on Open Source Systems Genova, 2005, pp. 273–278.
[12] M. Cataldo, J. D. Herbsleb, K. M. Carley, Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity, in: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, 2008, pp. 1–11. doi.org/10.1145/1414004.1414008
[13] D. Gremler, The Critical Incident Technique in Service Research, Journal of Service Research, 7 (2004) 65–89. doi:10.1177/1094670504266138
[14] E. Peer, L. Brandimarte, S. Samat, A. Acquisti, Beyond the Turk: Alternative platforms for crowdsourcing behavioral research, Journal of Experimental Social Psychology 70 (2017), 153–163. doi:10.1016/j.jesp.2017.01.006
[15] S. Palan, C. Schitter, Prolific.ac—A subject pool for online experiment, Journal of Behavioral and Experimental Finance 17 (2018) 22–27. doi:10.1016/j.jbef.2017.12.004
[16] M. Csikszentmihalyi, Beyond Boredom and Anxiety, Jossey-Bass Publishers, San Francisco, CA, 1975.
[17] S. Abuhamdeh, Investigating the "Flow" Experience: Key Conceptual and Operational Issues, Frontiers in Psychology 11 (2020). doi:10.3389/fpsyg.2020.00158
[18] J. Nakamura, M. Csikszentmihalyi, The concept of flow, in: Handbook of Positive Psychology (2002). doi:10.1002/9780470172698.ch19
[19] S. Engeser, N. Baumann, Fluctuation of Flow and Affect in Everyday Life: A Second Look at the Paradox of Work, Journal of Happiness Studies 17 (2016) 10–124. doi:10.1007/s10902-014-9586-4
[20] S. Abuhamdeh, M. Csikszentmihalyi, The Importance of Challenge for the Enjoyment of Intrinsically Motivated, Goal-Directed Activities, Personality and Social Psychology Bulletin 38 (2012). doi:10.1177/0146167211427147
[21] P. Sharafi, L. Hedman, H. Montgomery, Using information technology: Engagement modes, flow experience, and personality orientations, Computers in Human Behavior 22 (2006) 899–916. doi:10.1016/j.chb.2004.03.022
[22] A. Ghani, S. Deshpande, Task characteristics and the experience of optimal flow in human-computer interaction, The Journal of Psychology 128 (1994) 381–391. doi:10.1080/00223980.1994.9712742
[23] R. Agarwal, E. Karahanna, Time flies when you're having fun: Cognitive absorption and beliefs about information technology usage, MIS Quarterly 24 (2000) 665–694. doi:10.2307/3250951
[24] M. Tarafdar, C. L. Cooper, J.-F. Stich, The technostress trifecta - techno eustress, techno distress and design: Theoretical directions and an agenda for research, Information Systems Journal 29 (2019) 6–42. doi:10.1111/isj.12169

[25] J.-P. Mäkiniemi, S. Ahola, J. Joensuu, A novel construct to measure employees' technology-related experiences of well-being: Empirical validation of the techno-work engagement scale (technoWES) Scandinavian Journal of Work and Organizational Psychology 5 (2020) 1–14. doi:10.16993/SJWOP.79
[26] E. Demerouti, Job characteristics, flow, and performance: The moderating role of conscientiousness, Journal of Occupational Health Psychology 11 (2006) 266–280. doi:10.1037/1076-8998.11.3.266
[27] A. Bakker, The work-related flow inventory: Construction and initial validation of the WOLF, Journal of Vocational Behavior 72 (2008) 400–414. doi:10.1016/j.jvb.2007.11.007
[28] L. Ceja, J. Navarro, 'Suddenly I get into the zone': Examining discontinuities and nonlinear changes in flow experiences at work, Human Relations 65 (2012) 1101–1127. doi:10.1177/0018726712447116
[29] R. Quinn, Flow in Knowledge Work: High Performance Experience in the Design of National Security Technology, Administrative Science Quarterly 50 (2005) 610–641. doi:10.2189/asqu.50.4.610
[30] D. Taser, E. Aydin, A. O. Torgaloz, Y. Rofcanin, An examination of remote e-working and flow experience: The role of technostress and loneliness, Computers in Human Behavior 127 (2022) 107020. doi:10.1016/j.chb.2021.107020
[31] P. de Moura, N. de Oliveira Rosas, Perceptions about flow and boredom in the information technology profession: Evidence of a generational issue, International Journal of Human Capital and Information Technology Professionals 12 (2021) 1–17. doi:10.4018/IJHCITP.2021100101
[32] Z. S. H. Abad, M. Noaeen, D. Zowghi, B. H. Far, K. Barker, Two sides of the same coin: software developers' perceptions of task switching and task interruption, in: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering, EASE '18, ACM Press, New York, NY, USA, 2018, part F1377. doi:10.1145/3210459.3214170
[33] Z. S. H. Abad, O. Karras, K. Schneider, K. Barker, M. Bauer, Task interruption in software development projects: what makes some interruptions more disruptive than others?, in: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering, EASE '18, ACM Press, New York, NY, USA, 2018, pp. 122–132. doi:10.1145/3210459.3210471
[34] S. Graziotin, X. Wang, P. Abrahamsson, How do you feel, developer? An explanatory theory of the impact of affects on programming performance, PeerJ Computer Science 1 (2015) 1–14. doi:10.7717/peerj-cs.18
[35] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, What happens when software developers are (un)happy, Journal of Systems and Software 140 (2018) 32–47. doi:10.1145/3210459.3210471
[36] J. Flanagan, The Critical Incident Technique, Psychological Bulletin 51 (1954) 257–272. doi:10.1037/h0061470
[37] J. Gogan, M.-D. McLaughlin, D. Thomas, Critical incident technique in the basket, in: Proceedings of the 35th International Conference on Information Systems - Building a Better World Through Information Systems, ICIS '14, Association for Information Systems, 2014.
[38] M. Wirsing, J.-P. Banatre, M. Hölzl, A. Rauschmayer, Software-Intensive Systems and New Computing Paradigms - Challenges and Visions 5380 (2008). doi:10.1007/978-3-540-89437-7
[39] M. Kawabata, R. Evans, How to classify who experienced flow from who did not based on the flow state scale-2 scores: A pilot study of latent class factor analysis, Sport Psychologist 30 (2016) 267–275. doi:10.1123/tsp.2014-0053
[40] B. L. Berg, Qualitative Research Methods for the Social Sciences, 5th ed., Pearson Education, Boston, MA, 2004.
[41] H. Lune, B. L. Berg, Qualitative Research Methods for the Social Sciences, 9th ed., Pearson Education, Boston, MA, 2017.
[42] M. D. Meyers, Qualitative Research in Business and Management, 3rd ed., SAGE, London, 2020.
[43] P. de Moura, C. G. Porto Bellini, The measurement of flow and social flow at work: a 30-year systematic review of the literature, Personnel Review 49 (2020) 537–570. doi:10.1108/PR-07-2018-0240

[44] P. Armour, The learning edge, Communications of the ACM 49 (2006) 19–22. doi:10.1145/1132469.1132485
[45] J. Palomäki, T. Tammi, N. Lehtonen, N. Seittenranta, M. Laakasuo, S. Abuhamdeh, O. Lappi, B. U. Cowley, The link between flow and performance is moderated by task experience, Computers in Human Behavior 124 (2021) 106891. doi:10.1016/j.chb.2021.106891
[46] P. Graham, Maker's Schedule, Manager's schedule, 2009, URL: http://www.paulgraham.com/makersschedule.html
[47] F. Fagerholm, Software Developer Experience: Case Studies in Lean-Agile and Open Source Environments, Ph.D. thesis, University of Helsinki, Helsinki, Finland, 2015. http://urn.fi/URN:ISBN:978-951-51-1747-2
[48] P. B. Lowry J. D'Arcy, B. Hammer, G. D. Moody, "Cargo Cult" science in traditional organization and information systems survey research: A case for using nontraditional methods of data collection, including Mechanical Turk and online panels, Journal of Strategic Information Systems 25 (2016) 232–240. doi:10.1016/j.jsis.2016.06.002
[49] M. Salanova, A. M. Rodríguez-Sánchez, W. B. Schaufeli, Flowing together: a longitudinal study of collective efficacy and collective flow among workgroups, The Journal of Psychology 148 (2014) 435–455. doi:10.1080/00223980.2013.806290