How many pomodoros do professional engineers need to complete a microtask of programming?*

—Two Industrial Case Studies of Freelancers and Employees using Microtask Programming—

Shinobu Saito†
shinobu.saito@ntt.com
NTT Computer & Data Science Laboratories
Tokyo, Japan

Yukako Iimura
yukako.iimura@ntt.com
NTT Computer & Data Science Laboratories
Tokyo, Japan

Emad Aghayi
eaghayi@gmu.edu
George Mason University
Fiarfax, VA, USA

Thomas D. LaToza
tlatoza@gmu.edu
George Mason University
Fiarfax, VA, USA

Abstract

Microtask programming enables software engineers such as freelancers and part-time employees to contribute to software projects even when they can not spend much time on them. It decomposes software design into small, self-contained specifications. The decomposed specifications enable them to complete implementation and review task in a short time. In this paper, we empirically investigate the time required for software engineers to complete microtasks in an industrial setting and explore their perceptions of microtask programming by investigating two industrial projects using it. The projects were carried out in different companies and differed in the employment of the engineers. One contracted 9 freelancers, and the other asked for 8 part-time contributions from employees at work on other projects. We conducted a survey and a focus group with the engineers. Based on the development data of the case studies, we found that almost all microtasks were completed in less than four pomodoro repetitions, namely about two hours in the pomodoro technique. These data shows that engineers who cannot work full-time on a project can undertake microtasks if they can spare one-third of their work day. We also examine how engineers who are employees experience microtask programming similarly and differently from freelancers.

CCS Concepts

• **Software and its engineering** → *Application specific development environments.*

Keywords

Microtask, Programming, Specifications

*The plural of pomodoro in Italian is pomodori, but we describe it as pomodoros.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695263

ACM Reference Format:
Shinobu Saito, Yukako Iimura, Emad Aghayi, and Thomas D. LaToza. 2024. How many pomodoros do professional engineers need to complete a microtask of programming?: —Two Industrial Case Studies of Freelancers and Employees using Microtask Programming—. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3691620.3695263

1 Introduction

Engineers who work as freelancers and part-time employees cannot join projects full-time. By and large, such engineers want to work with flexible hours against a background of the rise of work style diversity. After the COVID-19 pandemic, it is reported that Work From Home (WFH) allowed engineers to work at different times than they did before [7], making it more difficult to work continuously at pre-determined hours to match the working hours of other members. Organizations need to offer ways for such engineers to contribute to projects even though they only use short, fragmented time for the projects.

Microtask programming [13] offers a potential solution for industrial projects to receive contributions from those who do not work full-time. Programming tasks are broken down into very short, discrete tasks: microtasks. To do so, the design of the software is decomposed into a set of small, self-contained specifications, each with a clear and independent objective. These decomposed specifications, which we call *micro-specifications* [16, 17], enable engineers to contribute without prior project knowledge. They can complete implementation and review tasks in a short time (i.e., microtasks). As each microtask can be completed in isolation from other ongoing work, microtasks do not require engineers to expend time coordinating with other engineers.

On the other hand, programming tasks are separated by makers' schedule, not managers' schedule [8]. The manager's task is separated by one hour, but the programming task should take much more time. No matter how much microtask programming is used, one hour is too short and inefficient for engineers to perform adequately. In an industrial development project, how many hours can programming microtasks be separated by? A well-known time management technique is the pomodoro technique [5]. It involves

setting a 25-minute timer for concentration on a specific task, followed by a 5-minute break. Each 25-minute interval is called as a "pomodoro," which means tomato in Italian. The technique recommends taking a long break after four pomodoro repetitions, namely 2 hours = 4 × (25 minutes + 5 minutes). Based on this, we believe that programming tasks can be separated into two-hour units.

There are two primary characteristics of microtask programming: 1) short-term work with minimally sufficient information and 2) individual work in constrained communication. Most professional engineers would not have had the opportunity to program in this way before. Microtask programming requires a shift in development style for participating software engineers. To be successful, industrial projects aiming to use microtask programming need to pay close attention to how its engineers perceive it.

Existing studies of microtask programming focus on its technical feasibility in an experimental setting [2, 4, 6, 11, 13, 14]. These studies have not empirically investigated the time required for professional engineers to complete microtasks in an industrial setting, nor closely examined their perceptions of microtask programming. In this industrial paper, we aim to answer the following research questions:

RQ1: Can professional engineers complete a microtask of programming in two hours (= four pomodolo repetitions)?

RQ2: How do professional engineers perceive working with minimally sufficient information in constrained communication via microtask programming?

RQ3: Does microtask programming affect the motivation of professional engineers?

To answer the questions, we examine two industrial projects which made use of microtask programming. The two were carried out by different companies and differed in the employment of engineers. One contracted 9 freelancers, while the other asked 8 company employees to work part-time. We conducted a survey and focus groups with 17 professional engineers (9 freelancers and 8 employees) about their experiences with microtask programming.

The remainder of this paper is organized as follows. Section 2 describes related work in microtask programming. Section 3 introduces microtask programming and a platform, which we developed, to manage microtask programming workflows and monitoring. Section 4 describes the context of two industrial projects which make use of microtask programming and analyzes investigate the completion time of microtasks. Section 5 explains the results of the survey and focus group with the participants. We identified the differences between two groups: freelancers and employees. Section 6 discusses the implications from the results of the case studies, and Section 7 concludes.

2 Related Work

Organizations traditionally employ employees, who each work during the same business hours during the time period of the project. This is in part because of project meetings: not only fixed, regular meetings (e.g., daily stand-up) but also ad-hoc meetings. These may require the participation of all members of a team. These situations might be characterized as meeting heavy [18]. For this reason, industrial projects may have limited work-time flexibility.

One approach to addressing this problem is microtask programming, which decontextualizes work to reduce the need for coordination and context. Microtasking is one of several models for crowdsourced software development [22]. Crowdsourcing has been applied to software engineering activities such as design [14], implementation [2, 4, 11, 13], and testing[6]. Commercial providers such as TopCoder, uTest, and UserTesting.com offer platforms for crowdsourcing software engineering. Studies of TopCoder have revealed that the number of contemporary projects, documentation, and crowd workers has a key impact on project quality [19].

Newcomers to software projects face onboarding barriers to install necessary tools, identify and download dependencies, configure the environment, understand the codebase, and identify a task. As a result, newcomers spend considerable effort onboarding. Programming environments may offer developers a preconfigured development environment to help reduce onboarding barriers. Microtask programming aims to further reduce onboarding barriers by decontextualizing work into microtasks. For example, Apparition provides a dedicated environment where engineers create microtasks for crowd workers to design and implement UI elements [11]. CodeOn provides an environment where developers can ask for help from other developers [4]. Studies of microtask programming have found that workers can be onboarded in less than one hour and complete microtasks in under five minutes [1, 2, 12]. However, companies currently have low awareness of these approaches and their potential to reduce onboarding barriers [15].

Based on the success of open source outside companies, companies have adopted inner-sourcing [3]. In an inner-source project, any employee within a company may contribute to the project [20]. Inner-sourcing provides new models for encouraging internal collaboration among developers within a company [10]. Inner-sourcing may increase development efficiency and code quality while decreasing development cycles [3]. However, there are no industrial case studies, other than our prior work [16], related to microtask programming in the context of inner-sourcing.

3 Microtask Programming

3.1 Two Types of Micro-Specifications

A microtask consisted of the implementation of a small, self-contained specification (i.e., micro-specification) or the review of an implementation microtask. Each microtask directly corresponds to a micro-specification. We define two types of micro-specifications: frontend and backend [16, 17].

Frontend micro-specifications specify how to build a user interface (i.e., system screens) while backend micro-specifications specify how to build backend logic (i.e., classes). To create frontend micro-specifications, the UI design images are decomposed into UI components and UI parts. Fig. 1 gives an overview of the relationships among UI parts, UI components, and system screens. Inspired by Atomic Design [9], UI parts are the basic building blocks of the UI (e.g., input boxes, buttons). UI components group several related UI parts. UI parts and UI components are together grouped into system screens. Fig. 2 shows an example of a frontend micro-specification for the Offer UI component. It contains two parts: parameters and

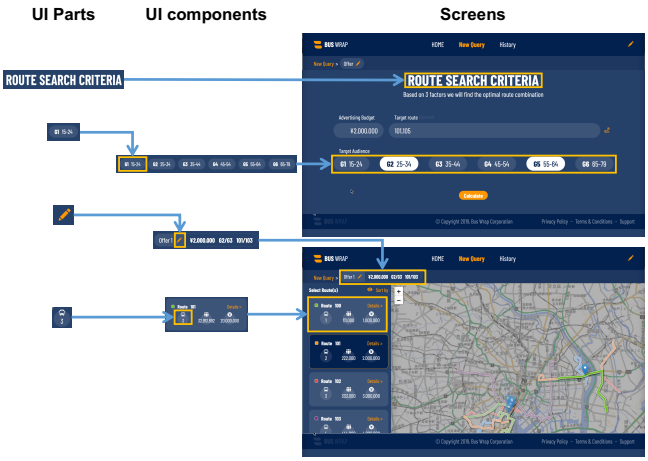


Figure 1: Screens are decomposed into UI components, which are decomposed into UI parts [17].

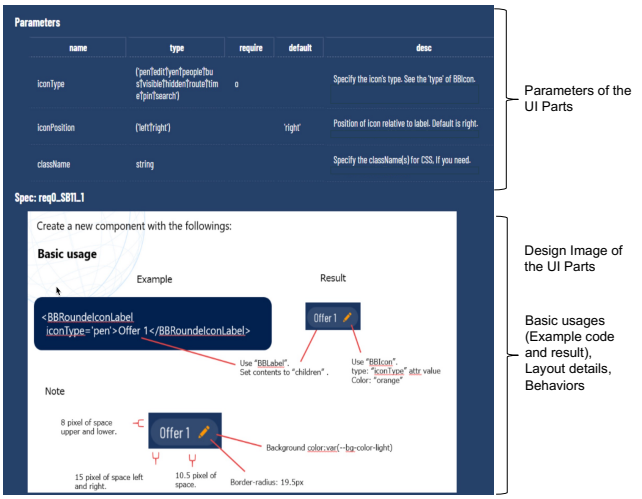


Figure 2: A micro-specification for Offer UI component

design images. By referring to a design image depicting the appearance of the user interface alongside a description, engineers may implement the UI component using the parameters.

An example of micro-specifications for the backend, as shown in Fig. 3. Typically, one method in a Java class corresponds to one micro specification. To begin work on a backend microtask to implement a part of a method, an engineer read a description of related abstract data types, an overview of the method's purpose, and micro-specifications describing its expected behaviors.

3.2 Two Types of Roles

Microtask programming cannot be used to develop all the source code of a system. As discussed in the later case study section, programming tasks for one part of the system will be carved out as microtasks. So, we defined two roles: *Microtask worker* and *Dedicated worker* [16, 17]. The former is not full-time engineer who uses

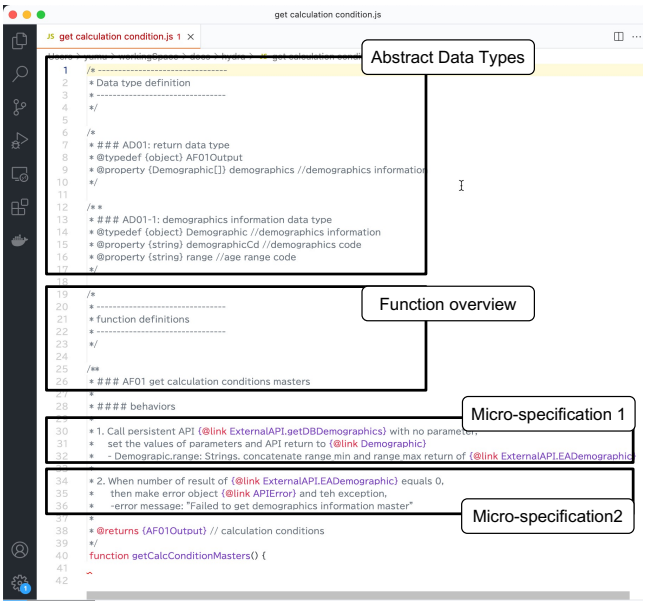


Figure 3: A micro-specification for the backend module.

available fragmented work time periods to complete microtasks. The latter is full-time engineer who is responsible for development tasks other than microtasks. They handle systems design and then create micro-specifications by decomposing design documents. Unlike microtask workers, the dedicated workers complete tasks that require more project knowledge such as the system architecture and database structure of the project.

3.3 Two Types of Constraints on Communication

To enable microtask workers to contribute whenever and wherever they want, coordination between workers is strictly constrained in microtask programming. The first constraint is that there are no meetings or synchronous, face-to-face communication between dedicated workers and microtask workers. For example, when a microtask worker asks a dedicated worker a question regarding a micro-specification, the microtask worker issues a new inquiry ticket via the ITS. A dedicated worker is then assigned the ticket and responds. However dedicated worker may sometimes want to communicate with all microtask workers immediately. To do so, dedicated workers may use tools such as a wiki or chat. The second constraint that microtask workers did not have an opportunity to communicate with other microtask workers, either synchronously or asynchronously. Moreover, microtask workers did not participate in any type of meetings during the whole period.

3.4 Two Types of Workflows and Platform

Since there are two types of microtasks, there are two workflows: implementation and review workflows. To manage the workflows, we developed a platform: Microtask WOrkflow and Reward calculation Platform (MWORP). It is a clinet-server application which consists of two key components: Microtask workflow automation

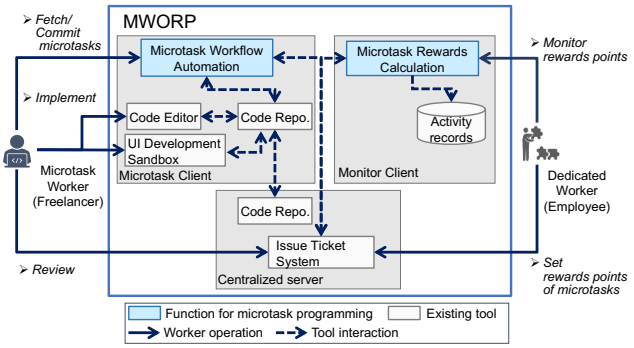


Figure 4: A platform, MWORP, provides workflow automation to fetch and automatically assigns microtasks to workers as well as calculates reward points based on the progress of them. Other functionality is provided through existing tools.

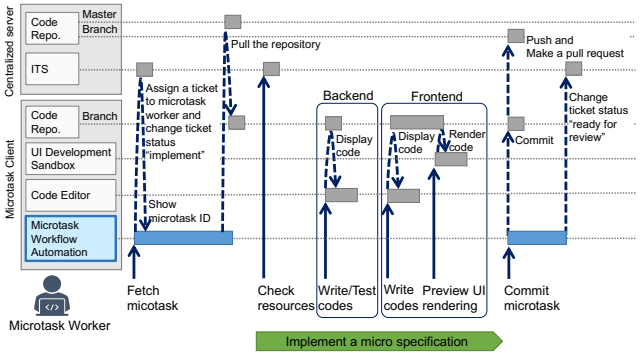


Figure 5: In the implementation workflow, microtask workers fetch a microtask, implement it, and commit it.

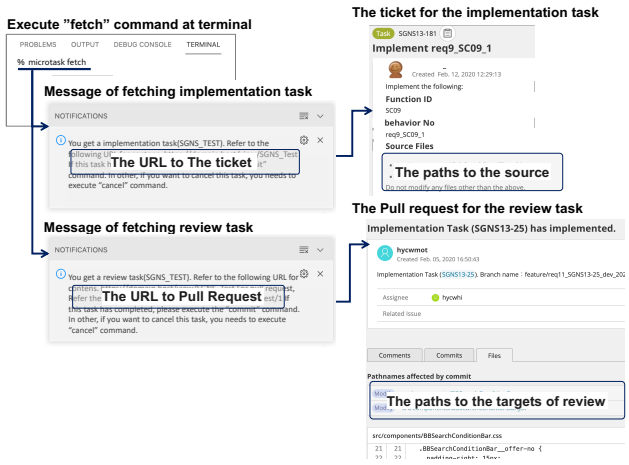


Figure 6: A screenshot of executing the "fetch" microtask command in MWORP.

and Microtask reward calculation (Fig. 4). Microtask workflow automation automatically manages workflows of two types of microtasks. Microtask reward calculation measures microtask workers' contributions to the project.

The centralized server of MWORP includes an issue tracking system (ITS) and a code repository of a version control system (VCS). The dedicated worker creates the micro-specifications and then uses the ITS to issue a ticket describing them. As each micro-specifications corresponds one-to-one to a microtask, tickets and microtasks are equivalent.

There are two types of MWORP clients: microtask clients and the monitor client. Microtask workers interact with microtask clients to fetch, implement, and commit their microtasks. Microtask clients include a local git repository, Microtask Workflow Automation (VS Code plugin), the code editor (VS Code), and the UI development sandbox (storybook.js). The monitor client enables dedicated workers to monitor the progress of the microtasks completed. MWORP is agnostic to the programming tools used by workers. The ITS and VCS are fixed by the platform due to API dependencies.

The MWORP microtask workflow automation assigns and manages microtasks automatically to help microtask workers focus on the implementation and review activities while minimizing distractions. This streamlines interactions with the ITS and the code repository by automating three processes: (1) fetching microtasks from the ticket pool in the ITS, (2) changing the ticket status based on the microtask progress, and (3) handling interactions between the local repository and the main repository in the VCS. As shown on the Fig. 6, they interact with the client through the command line in the terminal window of the code editor.

3.4.1 Implementation Workflow. Fig. 5 shows the implementation workflow. It consists of the following three steps:

Step 1. Fetch microtask: When microtask workers enter the "microtask fetch" command in the terminal as shown in Fig 6, the microtask workflow automation identifies a ticket whose current state is not closed, and assigns the ticket to the microtask worker. Then, the microtask client shows the URL of the ticket and identifier of the microtask in a pop up, pulls the remote repository in the cloud to the local repository on the microtask client, and creates a new develop branch.

Step 2. Implement the microtask: Mircotask workers write code and execute unit tests. Workers use VS Code for backend microtasks, and the Storybook UI development sandbox for frontend microtasks. When microtask workers want to stop work before completion, they may enter the "cancel" command. After a worker cancels a microtask, it will never again be assigned to them.

Step 3. Commit microtask: After the microtask worker enters the commit command, the develop branch is pushed to the remote repository, a pull request is created, and the ticket status in the ITS is updated.

3.4.2 Review Workflow. The review workflow has three steps:

Step 1. Fetch microtask: The interaction between microtask workers and the microtask workflow automation is the same as in the implementation workflow. MWORP shows the URL of the pull request. In the pull request, microtask workers find the path to the source code (Fig 6).

Table 1: Characteristics of Case Study 1 and 2

	Case 1	Case 2
Company		
Name	Company X	Company Y
Employees	About 3,000	About 2,000
Industry type	Telecom	IT service
Development Product		
Product type	B2B SPA	Bot Apps
Prog. language	JavaScript	TypeScript
Period of time		
Total construction time	7 weeks	5 weeks
Microtask prog. time	5 weeks	4 weeks
Project organization		
Microtask worker	8	9
Dedicated worker	2	3
Contract type		
Dedicated worker	Employment	Employment
Microtask worker	Freelancing	Employment
Micro-specifications		
for Frontend	27	-
for Backend	14	31
Total	41	31
Items of contributions		
Implemented by		
Microtask worker	35	31
Dedicated worker	6	0
Reviewed by		
Microtask worker	34	30
Dedicated worker	7	1
Development Size (LOC)		
by Microtask worker	3,184 (56%)	890 (71%)
by Dedicated worker	2,531 (44%)	357 (29%)
Total	5,715 (100%)	1,247 (100%)

Step 2. Review the implementation: When reviewing frontend microtasks, workers may also use the UI development sandbox to render UI images. After reviewing the implementation, workers decide to accept or reject the contribution. If the implementation is accepted, the worker merges the pull request to the main repository. If the implementation is rejected, the worker closes the pull request.
Step 3. Commit microtask: After the microtask worker enters the commit command, the ticket status is updated in the ITS.

4 Case Studies: Microtask Programming in Two Industrial Projects

We conducted two case studies of the use of microtask programming across two varying contexts. Table 1 lists the characteristics of each case. Each was conducted at a different company: X (Telecom) and Y (IT service). The cases differed in the employment arrangement of members using microtask programming. In Case 1, external freelancers were contracted while the company Y' employee were temporary assigned in Case 2. All project members in each project were different persons. None of the members had used microtask programming before.

Table 2: Microtask workers' development experiences

Years of Experimence	Freelancers in Case 1	Employees in Case 2
0–1 year	0 (0%)	0 (0%)
2–5 years	1 (12.5%)	0 (0%)
6–10 years	3 (37.5%)	0 (0%)
11–20 years	2 (25.0%)	3 (33.3%)
Above 21 years	2 (25.0%)	6 (66.7%)
Total	8 (100%)	9 (100%)

Table 3: Execution results of implementation microtasks

Result	Case 1 Frontend	Case 1 Backend	Case 2
Accepted	22 (66.7%)	13 (72.2%)	31 (53.4%)
Rejected	11 (33.3%)	5 (27.8%)	0 (0%)
Canceled	0 (0%)	0 (0%)	27 (46.6%)
Total	33 (100%)	18 (100%)	58 (100%)

4.1 Development Product and Period of Time

4.1.1 Case 1. Company X developed a B2B single-page application (SPA) for a bus company with two main features: bus route recommendations based on advertisements and interactive geographic mapping. The project was developed in JavaScript. The total period of software development was seven weeks. After creating a software design during a two week period, the project then implemented the design through microtask programming during a period of five weeks. This project had 10 geographically distributed contributors: 2 dedicated workers and 8 freelancers. The two dedicated workers were a requirements engineer and a software designer who were employees of Company X. They worked full-time on the project. The eight freelancers were recruited from outside Company X and had not previously worked with the company. Most (87.5%) of them had over six years of experience as described in Table 2. Freelancers were asked to make use of their fragmented time to work at their convenience.

4.1.2 Case 2. The project in company Y developed a bot application for their existing commercial product (business chat). It supports chat users by sending automated notifications for scheduled events. The bot was developed using TypeScript. After creating the software design documents for the product during a period of one week, the microtask programming phase then took place over a period of four weeks. Project members consisted of three dedicated workers and nine microtask workers who were employed by Company Y. All had over ten years of experience at it, as listed in Table 2. None had previously interacted before the project. Three dedicated workers worked full-time on the project while nine microtask workers had been assigned to different projects. So, microtask workers need to make use of their time to contribute to the project using microtask programming. Therefore, before the project started, the project manager asked microtask workers' report lines to allow them to use their fragmented work day time (9 am to 7 pm) to contribute to the project.

4.2 Microtask Workers's Compensation

Both projects assigned a point value, reward points, to each microtask to be completed. Points are set based on the contents of the microtask. After completing a microtask, the microtask worker receives the reward points. The compensation mechanisms in each case differed. In Case 1, freelancers received money based on their reward points. They were not paid a base salary. In Case 2, as the microtask workers were employed by the company, they were paid their regular salary. The reward points they received did not translate into any additional compensation.

4.3 Implemented Functionality

4.3.1 Case 1. The project developed both types of micro-specifications. It involved 41 micro-specifications, including 27 frontend and 14 backend micro-specifications. 8 microtask workers completed 85% of the implementation microtasks (35 of 41) and 83% of the review microtasks (34 of 41) while 2 dedicated workers completed the remainder. 5,715 lines of code were written. The dedicated workers wrote 44% of them, while microtask workers wrote about 56%.

Table 3 shows the results of microtask execution by the microtask workers. As mentioned above, they completed 35 implementation microtasks, of which 22 are for frontend and 13 are for backend. The frontend implementation microtasks were rejected 11 times by the review microtasks, while the backend implementation microtasks were rejected 5 times. This means that 33 (=22+11) frontend and 18 (=13+5) backend microtasks were finally executed.

4.3.2 Case 2. This project, unlike Case 1, involved only backend micro-specifications, as the product is a chatbot application. The bot had three features and consisted of 16 TypeScript classes. The classes were decomposed into backend micro-specifications. As in Case 1, one method in a class corresponds to one micro specification. The number of methods per class ranged from one to three. Overall, there were 31 backend micro-specifications for 31 methods across 16 classes. Nine microtask workers completed 100% of implementation microtasks (31 of 31) and 97% of review microtasks (30 of 31). 1,247 lines of code were written. About 29% were implemented by the dedicated workers and about 71% by the microtask workers.

Unlike Case 1, none of the microtasks were rejected in Case 2 as listed in Table 3. All 31 implementation microtasks were accepted by review microtasks. On the other hand, microtask workers in Case 2 cancelled microtasks a total of 27 times, once while starting them. Therefore, a total of 58 (=31+27) microtasks were executed.

4.4 Monitoring Microtask Progress

Reward points for microtask workers are calculated based on the results of microtasks to be completed. By using them, MWORP also enables dedicated workers to monitor project progress (i.e., check the microtask's status). Fig. 7b shows burn-up charts for each case created by the MWORP monitor client during the project period. Like a scrum master creating an agile release plan, the dedicated workers referred to these to track the progress of the microtasks they released and then decided when to release new microtasks.

Both projects did not have any meetings, including face-to-face or project progress meetings, during the whole project period. All microtask workers individually completed microtasks without the assistance of any other microtask workers or dedicated workers. They worked whenever and wherever they wanted. Fig. 8a and Fig. 8b show timelines of microtask workers' activities in Case 1 and Case 2, respectively. The Y axis represents the date and the X axis the time of day. Weekends and holidays are displayed in red. The distributions of working hours were different in each case. In Case 1, freelancers chose when to use their fragmented time to work, preferring to work early in the morning, late at night, or on weekends and not at a fixed time. In Case 2, employees were not allowed to work early morning or late at night by their company. They instead worked during working hours from 9 am to 7 pm. In both cases, workers were able to use their fragmented time to contribute to the project using microtask programming.

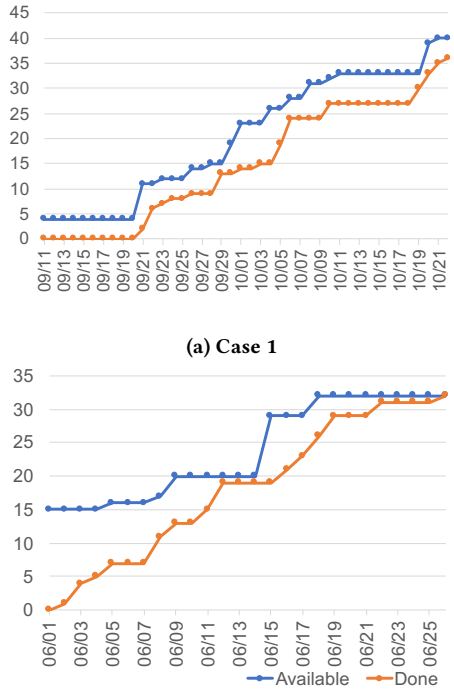
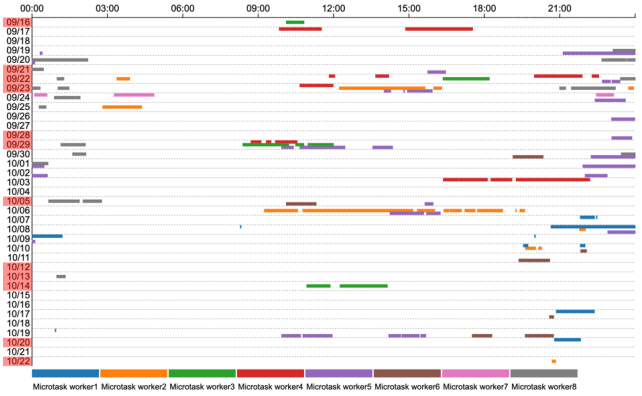


Figure 7: Burn-up charts of (a) Case 1 and (b) Case 2 used by dedicated workers to track the progress of microtasks released as well as plan the release of new microtasks.

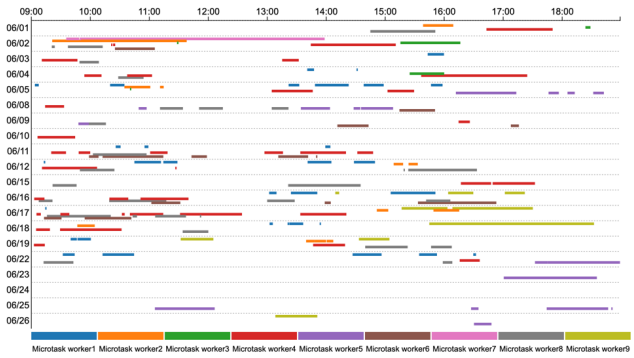
4.5 Elapsed Time of Microtasks

To measure the elapsed time for microtask completions in each case, we used the timestamps recorded in each ITS ticket within MWORP. Fig. 9 plots the distribution of elapsed time in each case. Overall, most microtasks of programming could be completed in 2 hours. Some of the implementation tasks exceed 2 hours, but almost all review microtasks are within 1.5 hours, even less than 2 hours.

4.5.1 Case 1. The left two violin plots show the completion times for frontend and backend microtasks in Case 1. The average completion time for the 33 frontend implementation microtasks was 83 minutes, and that for the 18 backend implementation microtasks



(a) Case 1



(b) Case 2

Figure 8: Timeline views of the fragmented work time of microtask workers in (a) Case 1 and (b) Case 2. Each bar plots a single microtask from fetch to completion, with the date on the Y axis (weekends and holidays highlighted in red) and time on the X axis. Each microtask worker is indicated with a distinct color.

was 31 minutes. On the frontend, 79% (=26/33) implementation microtasks and 100% (=33/33) review microtasks were completed within 2 hours. On the backend, 78% (=14/18) implementation microtasks and 94% (=16/17) review microtasks were completed within 2 ours.

4.5.2 Case 2. The right violin plot shows completion times in Case 2. The bot application was composed of only backend features. The average completion time was 74 minutes for implementation microtasks and 24 minutes for review microtasks. 81% (=47/58) implementation microtasks and 100% (=40/40) review microtasks were completed within 2 hours.

5 User Study: Exploring Perceptions and Motivations of 17 Engineers

To understand professional engineers' perceptions of microtask programming and changes in their motivations during the project period, we asked all freelancers and employees in the cases to complete a short questionnaire. We then conducted a group interview

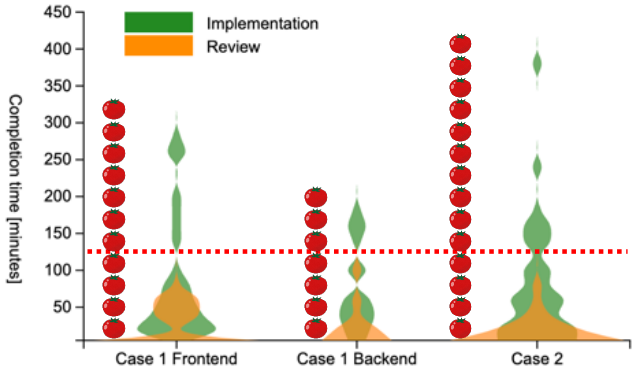


Figure 9: Distribution of completion times for implementation and review microtasks in Case 1 and Case 2. Most microtasks could be completed in four pomodoro repetitions, which is less than two hours. Some implementation microtasks exceeded 2 hours, but almost all review microtasks were within 1.5 hours, even less than 2 hours.

Table 4: Questionnaire

No.	Type	Question
Q-1	Likert scale	What do you think about working in a small unit?
Q-2		What do you think about working in asynchronous communication with designers?
Q-3		What do you think about with non-communication of co-workers?
Q-4	Yes/No	Did you maintain your motivation during the period of microtask programming?

in each group. In this user study section, we report the answers to the questionnaire and the opinions from the group interview.

5.1 Questionnaire Survey

We conducted a questionnaire survey of workers in both cases. We asked all freelancers and employees to evaluate the two primary characteristics of microtask programming: 1) short-term work with minimally sufficient information and 2) individual work in constrained communication. Survey questions also examined the impact of microtask programming on the motivation of workers. As shown in Table 4, the survey included four items, Q-1 to Q-4, evaluating three dimensions: difficulty, preference, and motivation.

5.1.1 Perceived difficulty and preference of short-term work with minimally sufficient information. Participants rated the level of difficulty of working in a short period of time with minimally sufficient information (Q-1: What do you think about working in a small unit?). Fig. 10 shows the results. Opinions varied between freelancers. Some felt that completing short tasks was easy, while others did not. A little less than 70% of employees reported difficulties with the short-term tasks.

5.1.2 Perceived difficulty and preference of individual work in constrained communications. We asked the participants to answer two

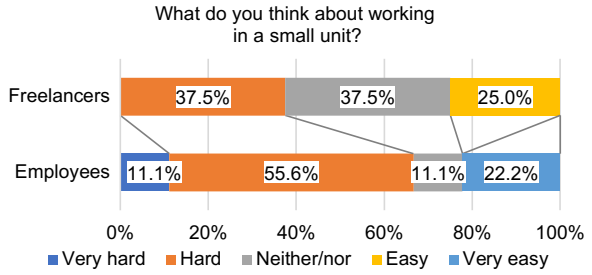


Figure 10: Perceptions of the difficulty of short-term work with minimally sufficient information (Q-1).

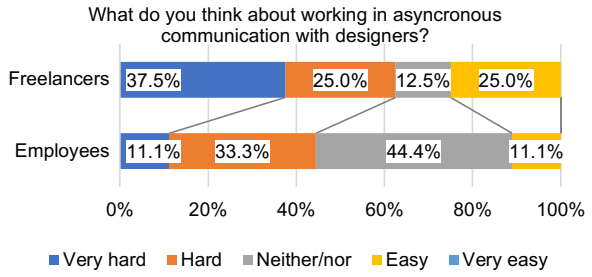


Figure 11: Perceptions of the difficulty of asynchronous communication with a dedicated software designer (Q-2).

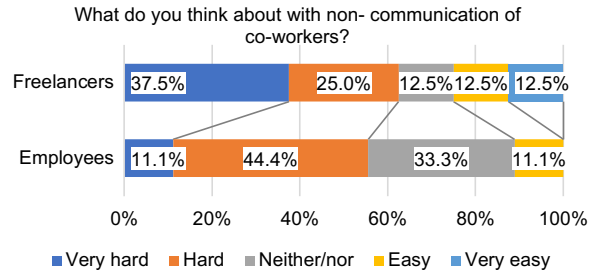


Figure 12: Perceptions of the restricted communication with co-workers (Q-3).

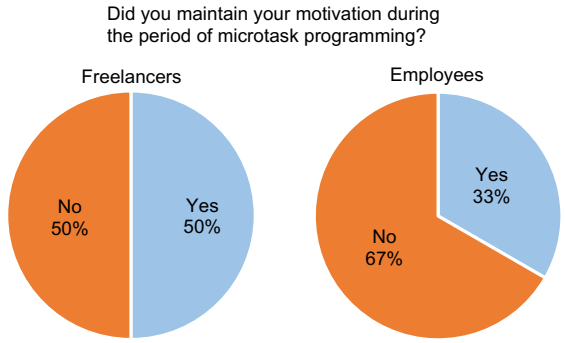


Figure 13: Motivation when microtask programming (Q-4).

questions about their experiences with communication in microtask programming. First, we asked them to rate their experience communicating with dedicated workers (Q-2: What do you think about working in asynchronous communication with designers?). Second, we asked them to rate their experience communicating with other microtask workers (Q-3: What do you think about with non-communication of co-workers?). As shown in Fig. 11, over 60% of freelancers reported that using only asynchronous communication with dedicated workers was inconvenient. About 45% of employees in Case 2 had difficulty doing that. Fig. 12 shows that over 60% of freelancers reported that having no communication with other workers was inconvenient. About 55% of employees reported difficulties without being able to easily communicate with other workers.

5.1.3 Impact on Motivation. We asked workers if they had stayed motivated (Q-4: Did you maintain your motivation during the period of microtask programming?). As shown in Fig. 13, motivation varied between freelancers and employees. Half of freelancers reported that they stayed motivated, while more than two-thirds of employees were demotivated.

5.2 Focus Group

We conducted a focus group for each case. A moderator asked questions about the two characteristics of microtask programming: short-term work with minimally sufficient information and individual work in constrained communication. The moderator invited them to discuss freely about the positive and negative aspects of the characteristics. Table 5 summarizes the perceptions of freelancers and employees.

5.2.1 Short-term work with minimally sufficient information. Both freelancers and employees had split perceptions of the impact of the short-term and minimally sufficient information nature of microtask programming on their work. Some felt that completing short tasks (i.e., microtasks) was a positive. For example, one employee reported that microtasks were helpful because they can make use of their small, available time whenever they choose to use it. Other freelancers reported a need for additional information about the project's status and the work of other members to work more efficiently and effectively.

5.2.2 Individual work with constrained communication. All freelancers reported that having no communication with other workers was inconvenient. For instance, one freelancer reported that they wanted face-to-face communication to address misunderstandings with other workers. Some employees recognized the benefits of not having any communication at all because they can use their time free from others.

6 Discussion

Programmers' tasks should take longer while managers' tasks are separated by one hour, as described in Section 1. From the results of the two case studies, most of the microtasks could be completed less

Table 5: Perceptions of Microtask Programming

Characteristic	Freelancers in Case 1	Employees in Case 2
1) Short-term with minimally sufficient information	***Positives***	***Positives***
	"I want to do small, easy tasks to complete a lot of tasks and want to make money."	"I was grateful that I didn't have to worry about where to do it."
	"Task size was small. So, I finished tasks [in a short time] even though I started to work after my children went to sleep."	"If I have one hour of available time, it would be nice to be assigned a microtask that can be completed in one hour."
	"I did not imagine the overview of the target system. I just did what I had to do. It's not important to know it."	"Short-temr work will be easy to engage in multiple projects. I will be happy to be involved in development if the project selects a [programming] language that I am more comfortable with."
	"I did not know the overview of that. But, it did not matter to me. I only focused on my scope."	
	"I was glad to get to experience a new technique in this small project."	
	Negatives	***Negatives***
	"When [micro-]specifications have ambiguous points, it's challenging for me to implement them."	"The size of the microtask is too small. I felt there were many split losses."
	"I don't know the system overview. So, I can't come up with ideas for improvements."	"If I had known the purpose and use cases of the app, I could have conducted unit tests using parameters that simulate actual operations."
	"I don't feel I create software products. I didn't feel a sense of accomplishment at all."	
2) Individual work in constrained communication	***Positives***	***Positives***
	None	"It was an advantage to be able to take time freely [without others interruptions]."
		"No need to coordinate participant schedules and I was able to work in my own free time."
	Negatives	***Negatives***
	"As usual, I often ask the software designer. I wish I asked [dedicated workers] by using chat and call."	"I couldn't get the information I was looking for due to a misunderstanding of the dedicated workers."
	"I think other workers misunderstood the specifications. F2F meeting is good for solving the misunderstanding. I wanted to share information with workers."	"Information is exchanged via a question ticket [with dedicated workers], I could not get an immediate answer to questions, even for small questions."
	"I wish I had a quick call with the software architect."	"Since I didn't know nearly enough about the people I was working with, I didn't feel a sense of unity as a project."
		"It would have been better to have everyone share their information with project members when there were problems."

than two hours. The pomodoro technique states that a typical full-time worker can complete three sets of four pomodoro repetitions in one day. This means that engineers who cannot work full-time can undertake most microtasks if they can spare one-third of a full-time engineer's time. This means that those who are not contributing full-time to a project can use microtask programming to contribute. This is a key finding for industries examining the potential to introduce microtask programming to their projects.

Answer to RQ1

Can professional engineers complete a microtask of programming in two hours (= four pomodolo repetitions)?

- ***Implementation microtasks***. About 80% of the implementation microtasks were completed in 2 hours (= 4 pomodoro repetitions).
- ***Review microtasks***. Almost all review microtasks were completed in 1.5 hours (= 3 pomodoro repetitions), even less than 2 hours.

Short-term work with minimally sufficient information, which is the first primary characteristic of microtask programming, enabled microtask workers to work in fragmented work times. In both cases, most periods of the time were less than two hours due to the small specifications. Workers were able to work whenever they wanted. In this way, microtask programming offered workers work-time flexibility. While they experienced the benefits of microtask programming, workers had split feelings about the use of short-term tasks. Some felt that less information is a positive while others felt it was a negative. Employees felt that dealing with the short-term nature of tasks was more difficult than freelancers. We also found that freelancers and employees had the same preferences regarding this characteristic.

The second primary characteristic of microtask programming is to work individually in constrained communications. Microtask workers communicated with dedicated workers only through issue tickets and they did not communicate with other microtask workers at all since self-contained specifications did not require them to communicate with others. Our examination indicates that regardless of contract type, nearly half of microtask workers felt that constrained communications were inconvenient even though each microtask was able to be completed in isolation from other ongoing work. However, we found that freelancers and employees had slightly different perceptions regarding their preferences for constrained communications.

Answer to RQ2

How do professional engineers perceive working with minimally sufficient information in constrained communication via microtask programming?

1) Short-term work with minimally sufficient information

- ***Difficulty***. Nearly 40% of freelancers felt that dealing with smaller tasks was hard. Over 60% of employees said it was very hard or hard.
- ***Preference***. In both groups, some liked short-term work, while others disliked it. Supporters liked short-term work because they felt it helped them make better use of their fragmented time by starting and finishing work at times convenient for them. Detractors felt that the information was too limited to effectively contribute to the project. Consequently, they felt no sense of accomplishment or satisfaction in their work.

2) Individual work in constrained communication

- ***Difficulty***. Over 60% of freelancers felt inconvenienced by the asynchronous communication with dedicated workers and lack of communication with co-workers. Around 50% of employees felt that this was inconvenient.
- ***Preference***. Almost all freelancers disliked the constrained nature of communications. They wanted more face-to-face communication with both dedicated workers and other microtask workers. In contrast, some employees appreciated the restricted communication, as it meant fewer interruptions such as chat or email messages from others and less need to coordinate minor tasks like meeting scheduling.

We found that some differences between contract types were observed in changes in their motivations through the experience of microtask programming. We identified similarities and differences in the experiences of freelancers and employees. Our examination reveals that this new approach still needs to improve the quantity and frequency of information provided to workers as well as the communication modalities available between members. At the same time, it is important to consider engineers' preferences for microtask programming to ensure those making use of it are motivated. Finding new ways to share additional information about project context and workers' contribution to the progress of the project may help improve motivation.

Answer to RQ3

Does microtask programming affect motivations of professional engineers?

- ***Freelancers***. 50% of freelancers reported not being motivated when working using microtask programming.
- ***Employees***. About 70% of employees reported not staying motivated during the project period.

One potential reason for the differences in motivation between the two types may be the differences in their compensation. In particular, freelancers were compensated based on the amount of work they completed, while employees were not. Future work should examine the impact of compensation and incentives on motivation.

6.1 Limitations and threats to validity

One potential threat to internal validity is that project participants' productivity and their perceptions of microtask programming might be affected by other factors, such as the situation of other projects they participated in during the case study period. For example, if they were particularly busy as a result of their participation in other projects, they might have had less available time to participate in their microtask programming project and then had a reduced focus on their microtask. Consequently, they might have had more negative reactions to microtask programming and report being more demotivated.

As with any case study, one potential threat to external validity is that we only analyzed two projects. However, these projects were conducted in different companies. Microtask workers in each project were contracted by two difference mechanisms: freelancing and employment. Another threat is that our platform (MWORP) requires a specific ITS and VCS. It uses only their essential functions of them, however. So it is technically feasible to use other ITS and VCS for MWORP. Moreover, it does not depend on specific programming tools. Microtask programming is not inherently domain-specific and would be relevant for other projects beyond developing applications such as SPA and bot.

Reliability is the ability to repeat a study and observe similar results [21]. To reinforce our study's reliability, we defined and documented the microtask programming workflows and managed a platform for supporting/automating them. By following the workflows and using the platform, other researchers or practitioners may replicate the case study in their context.

7 Conclusion

This industrial showcase paper examined two industrial projects and then empirically reveals the time required for a microtask of programming in the industrial setting. Those projects were carried out in different companies and differed in the types of contracts of the engineers using microtask programming (i.e., microtask workers). One contracted 9 freelancers, and the other asked 8 employees. Based on the two case studies, we found that the microtask workers could complete most microtasks in less than four pomodoro repetitions (i.e., two hours). We also evaluated the perceptions of the professional engineers in their use of microtask programming. Through a survey and focus group in each case, we found similarities and differences in how freelancers and employees perceive microtask programming.

While the focus of this study was exclusively on the implementation and review work in construction phase, we plan to apply the concept of microtasking to other software development phases such as software design and system testing.

Acknowledgments

The authors wish to thank M. Oda, K. Kataoka, and M. Yamane for their invaluable support with our case studies.

References

[1] E. Aghayi, "Large-Scale Microtask Programming," 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Dunedin, New Zealand, 2020, pp. 1-2.

[2] E. Aghayi, T. D. LaToza, P. Surendra, S. Abolghasemi, Crowdsourced Behavior-Driven Development, Journal of Systems and Software, Volume 171, 2021, 110840.

[3] M. Capraro and D. Riehle. 2016. Inner Source Definition, Benefits, and Challenges. ACM Comput. Surv. 49, 4, Article 67 (December 2017), 36 pages.

[4] Y. Chen, S. W. Lee, Y. Xie, Y. Yang, W. S. Lasecki, and S. Oney. 2017. Codeon: On-Demand Software Development Assistance. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17). Association for Computing Machinery, New York, NY, USA, 6220–6231.

[5] F. Cirillo, The Pomodoro Technique: The Acclaimed Time-Management System That Has Transformed How We Work, Crown Currency, 2018.

[6] A. Dwarakanath et al., "Crowd Build: A Methodology for Enterprise Software Development Using Crowdsourcing," 2015 IEEE/ACM 2nd International Workshop on CrowdSourcing in Software Engineering, Florence, Italy, 2015, pp. 8-14.

[7] D. Ford, M. -A. Storey, T. Zimmermann, C. Bird, S. Jaffe, C. Maddila, J. L. Butler, B. Houck, and N. Nagappan. 2021. A Tale of Two Cities: Software Developers Working from Home during the COVID-19 Pandemic. ACM Trans. Softw. Eng. Methodol. 31, 2, Article 27 (April 2022), 37 pages.

[8] P. Graham, Maker's Schedule, Manager's Schedule, https://www.paulgraham.com/makersschedule.html

[9] Atomic Design: https://atomicdesign.bradfrost.com/

[10] InnerSourceCommons: https://innersourcecommons.org

[11] R. Krosnick, "Creating Interactive User Interfaces by Demonstration using Crowdsourcing," 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Lisbon, Portugal, 2018, pp. 277-278.

[12] T. D. LaToza, A. Di Lecce, F. Ricci, W. B. Towne and A. van der Hoek, "Microtask Programming," in IEEE Transactions on Software Engineering, vol. 45, no. 11, pp. 1106-1124, 1 Nov. 2019.

[13] T. D. LaToza, "Crowdsourcing in Software Engineering: Models, Motivations, and Challenges," 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, QC, Canada, 2019, pp. 301-301.

[14] T. D. LaToza, M. Chen, L. Jiang, M. Zhao and A. v. d. Hoek, "Borrowing from the Crowd: A Study of Recombination in Software Design Competitions," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 2015, pp. 551-562.

[15] R. Prikladnicki, L. Machado, E. Carmel, and C. R. B. de Souza. 2014. Brazil software crowdsourcing: a first step in a multi-year study. In Proceedings of the 1st International Workshop on CrowdSourcing in Software Engineering (CSI-SE 2014). Association for Computing Machinery, New York, NY, USA, 1–4.

[16] S. Saito, Y. Iimura, E. Aghayi, and T. D. LaToza. 2020. Can microtask programming work in industry? In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). Association for Computing Machinery, New York, NY, USA, 1263–1273.

[17] S. Saito and Y. Iimura. 2020. Hybrid sourcing: novel combination of crowdsourcing and inner-sourcing for software developments. In Proceedings of the 15th International Conference on Global Software Engineering (ICGSE '20). Association for Computing Machinery, New York, NY, USA, 81–85.

[18] Scrum Myths: Scrum is "Meeting Heavy": https://www.scrum.org/resources/blog/scrum-myths-scrum-meeting-heavy

[19] K. -J. Stol and B. Fitzgerald. 2020. Two's company, three's a crowd: a case study of crowdsourcing software development. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). Association for Computing Machinery, New York, NY, USA, 187–198.

[20] K. -J. Stol and B. Fitzgerald, "Inner Source–Adopting Open Source Development Practices in Organizations: A Tutorial," in IEEE Software, vol. 32, no. 4, pp. 60-67, July-Aug. 2015.

[21] R. K Yin, Case Study Research: Design and Methods (Applied Social Research Methods) 2nd Edition, SAGE Publications, 1989.

[22] M. Zulfiqar, M. N. Malik and H. H. Khan, "Microtasking Activities in Crowdsourced Software Development: A Systematic Literature Review," in IEEE Access, vol. 10, pp. 24721-24737, 2022.