情報処理学会論文誌 Vol.63 No.4 999–1007 (Apr. 2022)

働き方の多様性を実現するための マイクロタスクプログラミングの実践と評価

飯村 結香子[1,a)] 斎藤 忍[1,b)]

受付日 2021年8月2日, 採録日 2022年1月11日

概要：テレワークの普及は，メンバが 1 つの場所に集まって働くことを前提としてきたソフトウェア開発プロジェクトに働く「場所」の柔軟性をもたらすきっかけとなった．筆者らが提案する「マイクロタスクプログラミング」は，働く「時間」の柔軟性を実現し，IT 人材を拡大に貢献すると考える．「マイクロタスクプログラミング」を採用した開発プロジェクトの実践から，多様で柔軟な働き方の実現可能性を示し，採用上の課題を議論する．

キーワード：ソフトウェア開発，マイクロタスクプログラミング，細切れ時間

Industrial Practice and Evaluation of Microtask Programming for Achieving Working Style Diversity

Yukako Iimura[1,a)] Shinobu Saito[1,b)]

Received: August 2, 2021, Accepted: January 11, 2022

Abstract: Traditionally, all members of the software development project need to work at a single workplace and in the same business hour. However, due to COVID-19, remote work is increasing exponentially. It can increase flexibility in the workplace for project members. In this paper, we report on a industrial practice and evaluation of microtask programming for increasing flexible business hours of the dispersed software engineers. On microtask programming, software engineers work with micro specifications which were organized into a set of microtasks, offering a set of short and self-contained descriptions. Our results from the practice offer initial evidence for the potential value of microtask programming in terms of new software development work style and its acceptability of project members.

Keywords: software development, microtask programming, fragmented time

1. はじめに

日本国内での IT 人材の不足が叫ばれて久しい．2015 年時点で約 17 万人の IT 人材が不足しており，それ以降も IT 人材は減少し続けている [9]．一方，IT へのニーズは増加し続け，IT 人材への需要は拡大し続けている．

IT 人材の不足は，機会損失と長時間労働の 2 つの問題の要因となる．IT ニーズに対し，その IT の開発に必要なメンバを確保できなければ，プロジェクトを開始できず，ビジネスの機会を逃す．プロジェクトを開始できたとしても，遂行に十分な人材を確保できなければ，プロジェクトメンバ 1 人あたりの作業量が増大し，長時間労働につながる．

IT 人材不足の原因の 1 つに，従来のソフトウェア開発プロジェクトにおける働き方の低い柔軟性があると筆者らは考える．従来のソフトウェア開発では，プロジェクトメンバに対し，プロジェクト全期間にわたり，フルタイム従事を求める．柔軟な「時間」での働き方を実現できれば，活用が難しかった 2 種類の IT 人材の活用が見込める．1 つ目は，介護や育児などにより，長時間勤務が難しかったり，勤務予定の変更が生じやすかったりする人の活用である．2 つ目は，企業内の人材のプロジェクトの終了から別

[1] 日本電信電話株式会社
NTT Corporation, Chiyoda, Tokyo 100–8116, Japan
[a)] yukako.iimura.vr@hco.ntt.co.jp
[b)] shinobu.saitou.cm@hco.ntt.co.jp

のプロジェクトの開始までの限定的な期間や，業務時間内の一部を主業務とは別の業務に割り当てる限定的な時間の活用である．前者はプロジェクトで活躍する人材の枠を広げ，後者は企業内人材の流動性を高める．

筆者らはソフトウェア仕様を小さい部分（マイクロ仕様）に分割し，断続的かつ短時間の作業（マイクロタスク）を複数人が独立に実行するソフトウェア開発アプローチ「マイクロタスクプログラミング」を検討しており，本アプローチの企業での実現可能性に関する実践事例を報告している [8], [19]．マイクロタスクプログラミングは，ソフトウェア開発の実装工程に柔軟な「時間」での働き方を実現できるとの仮説により，次の研究設問（RQ）を設定する．

RQ：ソフトウェア開発アプローチ「マイクロタスクプログラミング」は，柔軟な働き方の実現に貢献するのか？

本稿の構成を以下に示す．2 章では柔軟な働き方のスタイル「細切れ時間ワーキングスタイル」を定義し，実践上の課題を示す．3 章では開発アプローチ「マイクロタスクプログラミング」を解説し，実践上の課題の解決について検討する．4 章では実践例を示し，5 章で評価，6 章で考察を行う．7 章で関連研究を述べ，8 章で結論を述べる．

2. 「時間」柔軟な働き方の定義と実現への課題

2.1 細切れ時間ワーキングスタイル

「時間」柔軟な働き方の実現には，ソフトウェア開発における 2 種類の「時間」制約の排除が必要である．1) プロジェクト参画期間と 2) 参画期間中の作業時間長・タイミングの制約である．1) では，プロジェクト全期間にわたる参画を求めず，中途からの参画や中途での離脱を許容する．2) では定常的かつフルタイムの参画を求めず，非定常的な短時間の作業を許容する．また，1)，2) いずれについても，事前に参画の期間，時間やタイミングを確定しないことを前提にする．

本稿では，「細切れ時間ワーキングスタイル」を，エンジニア自身がソフトウェア開発プロジェクトへの参画期間と期間中の作業時間長や作業タイミングを決定する働き方と定義する．また，その決定は事前に行われるものではなく，適宜変更可能とする．そのうえで，「細切れ時間ワーキングスタイル」のメンバと従来型のフルタイムメンバ（以降，専任開発者と呼ぶ）との組合せによりプロジェクトを遂行の実現を目指す．

2.2 細切れ時間ワーキングスタイル実践上の課題

2.2.1 「細切れ時間ワーキングスタイル」で実行可能なタスク特性

「細切れ時間ワーキングスタイル」を選択するエンジニアらの，「ソフトウェア開発プロジェクトへの参画期間と期間中の作業時間長および作業タイミングの決定」を実現するためにタスクが持つべき特性を整理する．

- **課題 A1「タスク着手に求められる初期参画コストが低いこと」：**プロジェクト途中の人員の追加は “新たに投入された開発者が生産性の向上に貢献するまでには，時間がかかる．新たにプロジェクトに参画した人は，仕事に取りかかる前に，まず開発の現状や設計の詳細などを理解しなければならない” といわれる [7]．「開発の現状や設計の詳細」などのプロジェクト知識の習得時間を初期参画コストと呼ぶ．長期間，定常的なプロジェクト参画では，総作業時間に比して初期参画コストは小さくなるが，細切れ時間ワーキングスタイルでは，プロジェクトの参画が短期間になる可能性が高く，より初期参画コストを低く抑える必要がある．
- **課題 A2「タスク単位が小さいこと」：**細切れ時間ワーキングスタイルのメンバは，作業時間長を自身で決定する．エンジニアが希望する時間長内にタスクが完了できるよう，タスクは短時間で実行できること，つまりタスク単位が小さい必要がある．タスク単位が小さいことは，着手したタスクが完了しなかった場合のリカバリコスト低減にもつながる．本稿では，エンジニアが実行するタスク単位は 1.5 時間程度の作業で完了できることを目安とした．中断を挟んだ複数回の作業による完了も許容する．
- **課題 A3「独立したタスク実行が可能なこと」：**細切れ時間ワーキングスタイルでは，エンジニアが自身の作業タイミングを決定する．また，その決定は事前には行わない．これは，定例会議などのエンジニア同士の同期的協調を前提とできないことを意味する．エンジニアが希望する任意のタイミングで，その時点で実行可能なタスクをエンジニアに割り当て，また，タスクの実行にも他者との協調が不要なことが求められる．

2.2.2 プロジェクトマネジメント視点の懸念

提案コンセプトについて，ソフトウェア開発を実践されている現場の方と意見交換をした際に，プロジェクトマネジメント視点から次の 2 つの懸念があげられた．これらの懸念に対し実態に基づく説明できることが必要である．

- **懸念 B1「小さなタスク実行はエンジニアのモチベーションを低下させないか」：**細切れ時間ワーキングスタイルの業務やシステムの概要情報などを省いた小さな単位の仕様に基づく作業に対し，エンジニアらが拒否感を抱くのではないか？という懸念があった．限定的な情報開示や詳細すぎる作業指示と感じ，モチベーションを失うのではないかという意見であった．
- **懸念 B2「専任開発者の負担が大きくなりすぎないか」：**細切れ時間ワーキングスタイルのメンバに作業をしてもらうための専任開発者の負担に対する懸念があった．専任開発者の作業内容や時間が，従来のソフトウェア開発プロジェクトとは著しく異なるのではないかという意見であった．

3. マイクロタスクプログラミング

筆者らは，「マイクロタスクプログラミング」によるソフトウェア開発アプローチの採用が，2章の「細切れ時間ワーキングスタイル」で実行可能なタスク特性を満たし，時間柔軟な働き方の実現に寄与すると考えた．「マイクロタスクプログラミング」ついて解説し，「細切れ時間ワーキングスタイル」で実行可能なタスク特性を満たすと考える理由を示す．

3.1 マイクロタスクと2つの役割

マイクロタスクプログラミングは，設計工程で専任開発者がソフトウェア仕様を小さい部分（マイクロ仕様）に分割し，実装工程では，マイクロタスカーが，それぞれのマイクロ仕様を実現するタスクを複数人で独立に実行することでソフトウェアを開発するアプローチである [19]．

- **専任開発者**：対象業務・システムおよびプロジェクトの全体を把握し，プロジェクト期間を通し，定常的に活動する役割．
- **マイクロタスカー**：プロジェクトへの参画期間を宣言し，参画期間中は任意のタイミングで断続的，短時間に作業する．細切れ時間ワーキングスタイルメンバは，マイクロタスカーとしてプロジェクトに参画する．
- **マイクロタスク**：マイクロタスカーが実行する，プロジェクト知識なしに，独立で完了できる小さなタスク．

3.2 技術的な特徴

3.2.1 特徴1「開発作業の分割」

ソフトウェア開発に関わる作業を2段階に分割をする．

- **ステップ1. 必要とする知識で作業を分ける** ソフトウェア開発作業を，対象となる業務やシステム全体の知識（プロジェクト知識）が必要な部分と不要な部分に分割する．機能の洗い出しや全体設計にはプロジェクト知識が必要だが，各モジュールの実装にはプロジェクト知識は不要である．専任開発者が前者を担当し，マイクロタスカーは後者を担当する．これにより，課題A1「タスク着手に求められる初期参画コストが低いこと」が実現できると考えられる．
- **ステップ2. 作業単位を小さくする** 前ステップで分割された，プロジェクト知識が不要な部分を，さらに他の作業に依存しない，小さな単位に分割する．分割後の小さな単位の作業をマイクロタスクと呼ぶ．

3.2.2 特徴2「仕様の分割と割当て指針」

一般的にソフトウェアは，複数のモジュールで構成される．マイクロタスクプログラミングではさらに，モジュール内をさらに複数の小さな部分（マイクロ仕様）に分割し，1つの仕様の実現を1つのタスクとする．マイクロ仕様については，実装されたか否かが，単体テスト（動作確認）可能な単位であることとする．現時点でモジュールの分割・モジュール内分割ともに特定の手法を定めてはいない．**図1**に仕様の分割とタスクの割当ての例を示す．例では，1つのモジュールをさらにマイクロ仕様1とマイクロ仕様2の2つに分割している．各マイクロ仕様に対し，実装とレビューの2種類のマイクロタスクが実行される．したがって，例では4つのマイクロタスクが実行される．あるマイクロ仕様の実装とレビューの2つのマイクロタスクは，異なるメンバが実行するよう割当てを行う．マイクロタスクを単位とする作業の割当ては，課題A2「タスク単位が小さいこと」を実現すると考えられる．

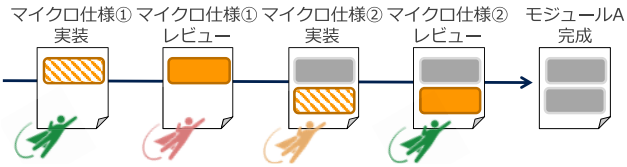

図1 仕様の分割と割当てのイメージ
Fig. 1 Microtask programming workflow.

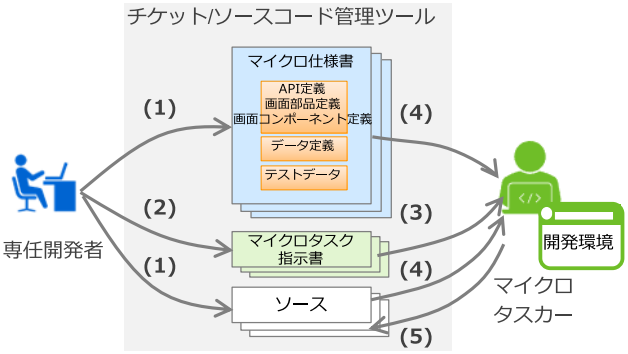

図2 役割間のモノ・情報の流れ
Fig. 2 Communication flow.

3.3 特徴3「非同期コミュニケーション」

専任開発者とマイクロタスカーのモノと情報の流れを**図2**に示す．両者のコミュニケーションは，チケット管理ツールとソースコード管理ツールを介して行う．専任開発者とマイクロタスカーの作業手順を以下に記す．

(1) 専任開発者は，全体設計，マイクロ仕様設計を行いマイクロ仕様書を作成，ソースコードテンプレートなど開発資材とともにソースコード管理システムに登録する．
(2) 専任開発者は，マイクロ仕様に対応するタスク指示書を作成し，チケット管理システムに登録する．
(3) マイクロタスカーは，任意のタイミングでチケット管理ツールにアクセスし，その時点で実行可能なマイクロタスク指示書を取得する．
(4) マイクロタスカーは，取得したマイクロ仕様に基づき，ソースコード管理システムより必要な開発資材を取得し，マイクロタスクを実行する．

(5) マイクロタスカーは，マイクロタスク完了後，成果物をソースコード管理ツールにコミットする．

以上の作業手順において，マイクロタスカーは他のメンバとのコミュニケーションや協調を必要としない．つまり，課題 A3「独立したタスク実行が可能なこと」を実現できると考えられる．

マイクロタスカーは専用開発環境で作業する．開発環境は，タスク割当てや作業結果のコミット時にチケット管理ツールのチケット更新を自動で行い，タスクがユーザに割り当てられてから，完了またはキャンセルされるまでの，タスクの開始/終了時刻，作業の中断/再開時刻を活動実況ログとして記録する．

4. 実践

4.1 実践に至る背景

実践した企業では，育児，介護などのライフイベントやリモートワーク，短時間・分断勤務などワークスタイルの多様化により，同期コミュニケーション/ミーティングを前提としたソフトウェア開発プロジェクトのみでは，社員の参画が難しいケースが増えてくるとの認識があった．ソフトウェア開発プロジェクトに細切れ時間ワーキングスタイルを採用できるのであれば，社員のスキル・知識を発揮する機会を拡大できるのではないかと期待を持った．また，プロジェクトとプロジェクトの狭間期間の活用や，シニア技術者らが新たなプログラミング言語などの知識やスキルを身につけながら小さな単位でプロジェクトに参画するなどの実践的なリカレント教育の可能性も期待された．そこで，細切れ時間ワーキングスタイルの実現可能性の評価，またプロジェクトマネジメント上への懸念に対する実態の把握を目的に，実プロジェクトへのマイクロタスクプログラミングの適用による実証実験を行うことにした．

4.2 開発プロジェクトの体制

プロジェクトは，専任開発者 3 名，細切れ時間ワーキングスタイルを選択するメンバ，つまりマイクロタスカー 9 名の体制で遂行された．マイクロタスカーとしてプロジェクトに従事するメンバは，社内で実証実験の趣旨および参画条件を公開し，希望者を募った（**図 3**）．

筆者らの想定する細切れ時間ワーキングスタイルを選択する状況には，育児，介護や自身の療養などにより，もっぱら細切れ時間ワーキングスタイルのみで仕事に従事するケースと，主業務は別にあり，副業務として細切れ時間ワーキングスタイルで従事するケースが想定される．本プロジェクトでは，後者を想定し，勤務日の 9 時～19 時の範囲でタスクを行うとした．参加期間は 3 週間程度，週あたりの参加時間は 6 時間程度を参加の目安とした．ただし，実際の作業量や時間帯は，主業務の状況他により各マイクロタスカーが決定できるものとした．

「実証実験への参加者募集」

「細切れ時間ワーキングスタイル」は，フルタイム勤務が難しい社員に活躍の機会を与え，育休等時短勤務からの復帰など，当社の開発現場で有効に活用できる可能性があります．実験への協力をお願いします

1～2 時間を想定したチケットに取り組んでください．作業量は，週に 6 時間程度，3 週間を目安とします．作業時間は定めません．自由な時間に参加下さい．開発プロダクト仕様の読込みなど事前作業は不要です．

必要スキル：REST API クライアント開発，Type Script，HTML5/Web アプリ，社内プロダクト API の基礎知識．

図 3 参加者募集の案内文
Fig. 3 Participant recruitment for case study.

専任開発者とマイクロタスカーは，同じ会社に属するが，同じプロジェクトに従事した経験などはなく，本プロジェクトの開始時にも，顔合わせなどは行わなかった．

4.3 開発プロダクト

開発プロダクトは，同社が保有する既存プロダクトを活用した新サービスの技術およびコンセプト検証を行う PoC（Proof of Concept）である．プロダクトは大きく 3 つの機能を持ち，16 クラスから構成される．クラス中の 1 メソッドの動作仕様を 1 マイクロ仕様とした．各クラスは，1 から 3 のメソッドを備える．全体で 32 メソッド＝32 マイクロ仕様が準備された．1 つのマイクロ仕様に対し，実装とレビュー 2 種のタスクを実行するため，マイクロタスク数は 64 であった．開発言語は Type Script である．

4.3.1 開発プロジェクト

本実証実験の対象となったのは，新規開発の設計から結合テストまでの工程である．機能概要は，専任開発者を担当したエンジニアらが実験以前より検討していた．本実証実験の開始にあたって仕様を確定し，マイクロ仕様設計を行った．マイクロ仕様書，マイクロタスク指示書および開発資材の登録は 3 日間で行われた．マイクロタスカーによるマイクロ仕様の実装 & Unit Test/レビューは約 1 カ月で行われた．同期間に専任開発者は，マイクロタスカーからの問合せ対応，またマージ作業を行った．すべてのマイクロタスクの完了後に結合テストが行われた．

期間中のマイクロ仕様の公開数，完了数の遷移を**図 4** に示す．マイクロ仕様の公開数が折れ線で，完了数を棒グラフで示す．灰色がけは週末（作業不可日）を表す．マイクロ仕様は段階的に公開されたため，公開数を示す折れ線は右上方向へとのぼる階段状となっている．

専任開発者およびマイクロタスカーの作業の結果，予定機能はすべて実現された．コードの約 7 割（Line of Code ベース）がマイクロタスカーにより実装された．マージ作業および結合テスト時に 2 つの動作不良が発見されたが，

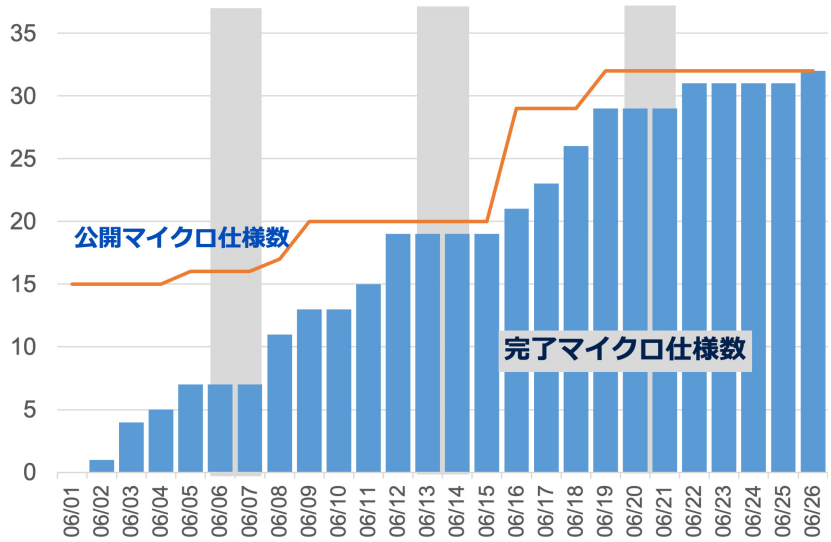

図 4 マイクロ仕様の公開数と完了数の遷移

Fig. 4 Burn up chart showing the number of transition of micro specifications assigned to microtask workers and the number of micro-specificatoins developed by them.

専任開発者による設定ファイルの記述ミスおよび動作仕様の曖昧さに起因するもので，マイクロタスクプログラミングおよび細切れ時間ワーキングスタイルにの採用により発生したものではなかった．

5. 評価

RQ：ソフトウェア開発アプローチ「マイクロタスクプログラミング」は，柔軟な働き方の実現に貢献するのか？について，「細切れ時間ワーキングスタイル」の実現に必要なタスク特性の 3 つの課題の達成状況と，プロジェクトマネジメント視点からの 2 つの懸念への実態を評価する．

5.1 課題 A1「タスク着手に求められる初期参画コストが低いこと」の達成状況

プロジェクト終了後，マイクロタスカーらに，通常プロジェクトへの途中参画と比した本プロジェクトへの参画の時間・手間について「とても小さい」から「とても大きい」の 5 段階評価で尋ねた（**表 1**）．全員が初期参画コストはとても小さい，または，やや小さいと回答した．従来の開発プロジェクトより初期参画コストが低い理由として，参照すべき情報量が少ないこと，参照先が明確であることをあげられた．実際にかかった時間は，30 分〜1 時間半が 6 名，半日程度が 2 名，残り 1 名が 1 日半であった．大多数が半日以下で最初のタスクに着手できており，課題 1 は達成されたといえる．要した時間は，開発環境の操作および作業手順の把握のためにかけられている．作業着手までに，最も長く 1 日半を要した人は，ふだん利用しているブラウザのプラグインが，本実証実験指定の開発環境では正常に動作しなかったため，設定の変更などが必要であった．

表 1 本プロジェクトへの参画の時間・手間は，通常プロジェクトに途中参画する時間・手間と比較してどうであったか？

Table 1 Questionnaire result about onboarding.

とても大きい	やや大きい	変わらない	やや小さい	とても小さい
0	0	0	5	4

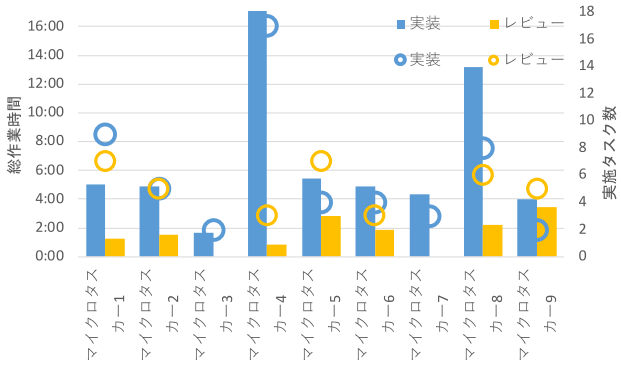

図 5 エンジニアごとの総作業時間，マイクロタスク数

Fig. 5 Total working time and Number of microtasks.

5.2 課題 A2「タスク単位が小さいこと」の達成状況

マイクロタスカーごとの総作業時間と実行タスク数（**図 5**）から，タスク単位が小さかったかを評価する．棒グラフが実装/Unit Test とレビューそれぞれの総合作業時間を，丸が実装/Unit Test とレビューそれぞれの実行タスク数を示している．エンジニアによりややばらつきがあるが，タスクあたりの作業時間は実装作業が平均で 1 時間 15 分，レビュータスクでは 25 分であった．マイクロタスカーの作業時間長の目安の 1 時間半を下回っており，タスク単位が小さかったと評価できる．

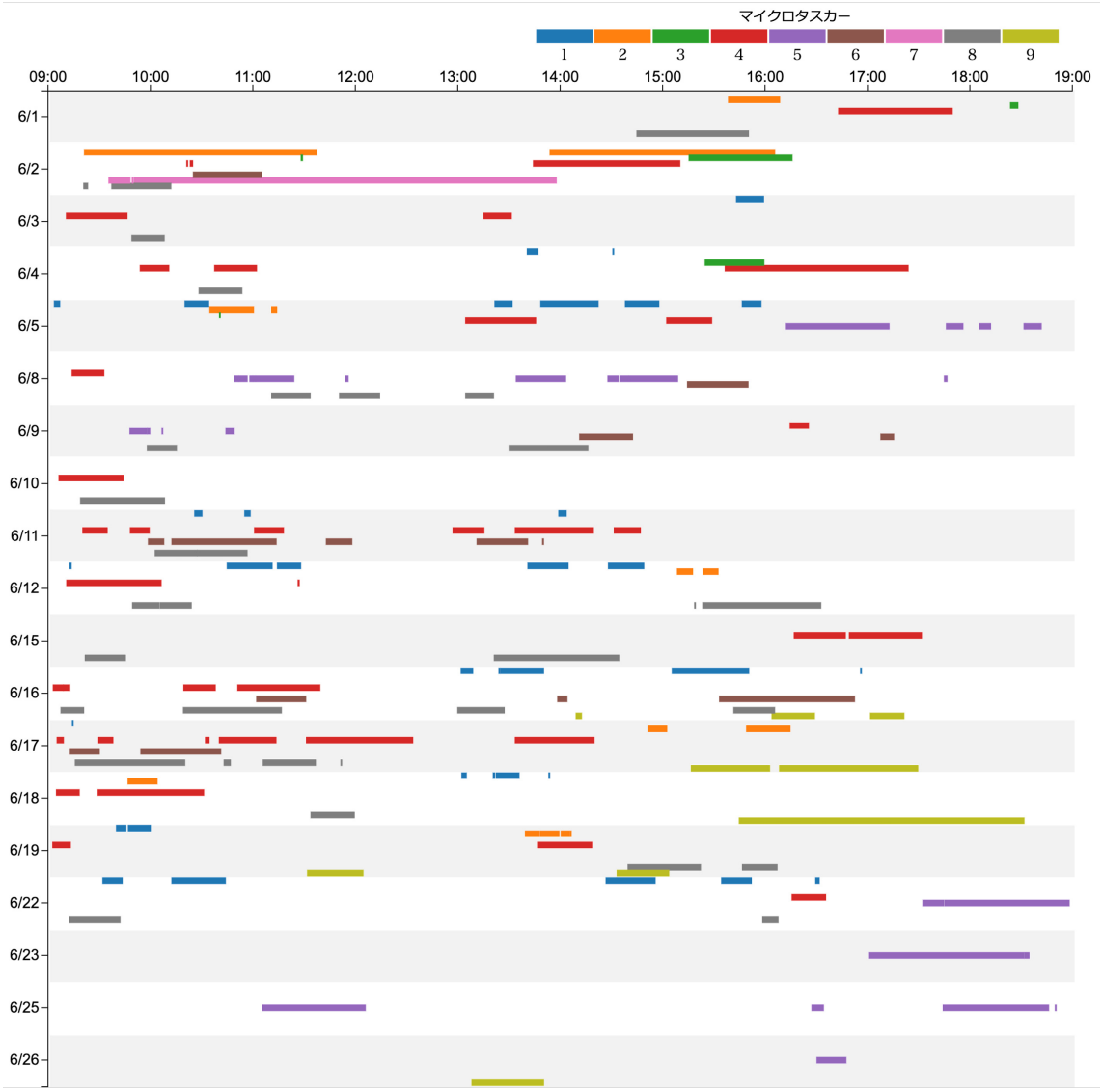

図 6 マイクロタスカーの作業時間帯
Fig. 6 microtask workers' working-time.

5.3 課題 A3「独立したタスク実行が可能なこと」の達成状況

本プロジェクトでは，定例の打ち合わせなどはいっさい実施していない．事後アンケートで，マイクロタスカーらに作業時間をどのように決定したかを自由記述で尋ねたところ，「本業務の合間に 30 分から 1 時間程度時間が取れそうなタイミングで実行した」，あるいは，「あらかじめ作業予定の曜日や時間帯を自分で設定し，実際に作業可能であれば実行した」との回答が得られた．

実際のマイクロタスカーの作業時間帯を活動状況ログの可視化を**図 6** に示す．Y 軸は作業日，X 軸は 9 時から 19 時までの作業可能時間帯である．矩形の左端位置が作業開始時刻を，右端が位置を作業終了時刻，矩形幅が作業時間長を示す．矩形色が作業を行ったマイクロタスカーを示す．マイクロタスカーらの作業時間帯（矩形色の出現位置）に，同期的な傾向は見られず，独立してタスクが実行されたことが分かる．また，マイクロタスカーごとにも一定のパターンを見い出すことはできない．アンケートの「本業の合間」や「作業が実行可能なタイミング」でタスクを実行したことが裏付けられる．

5.4 懸念 B1「小さなタスク実行はエンジニアのモチベーションを低下させないか」の実態

実証実験前には，マイクロタスクプログラミングはエンジニアのモチベーションを低下させるのではという懸念された．プロジェクト知識に対し多くの問合せが発生することや，マイクロ仕様単位での作業に対しエンジニアが拒否感を持つのではないかとの意見があった．実際には，マイクロタスカーが作業期間中に情報の公開を求めることやプロジェクト知識に対する質問はなかった．プロジェクト終了後のアンケートで「同じやり方のプロジェクトへ次回以降も参画したいか」を問うたところ，とても参画したくな

表 2 同じやり方のプロジェクトへ次回以降も参画したいか？
Table 2 Questionnaire result about willingness.

とても参画したくない	やや参画したくない	どちらでもない	やや参画したい	とても参画したい
1	1	3	2	2

いからとても参画したいの5段階でとても参画したくない，やや参画したくないと回答したのはそれぞれ1名であった（表 2）．

5.5 懸念 B2「専任開発者の負担が大きくなりすぎないか」の実態

実証実験前には，マイクロタスカーがプロジェクト知識を持たないこと，同期的なコミュニケーションがとれないこと，などから，専任開発者のマイクロ仕様の作成などの作業について，一般のソフトウェア開発と比して作業量や質が著しく異なることが懸念された．実態としては「マイクロタスク依頼のために記述するドキュメントの内容・量が，通常の開発と比べて極端に多いわけではなかった」，「想定よりゆるく書いた仕様で依頼したが，マイクロタスカーが思ったより意図を汲んで作業をしてくれた」と専任開発者らはプロジェクト終了後に行われたインタビューで回答している．マイクロ仕様の分割についても，「モジュール分割は一般的な開発と同じ．プロジェクト初期にモジュール内の分割は，関数単位と決定した後では悩むことはなかった」としている．

6. 考察

6.1 マイクロタスクプログラミングによる細切れ時間ワーキングスタイルの実現可能性

実証実験では，予定機能はすべて実装され，細切れ時間ワーキングスタイルおよびマイクロタスクプログラミングに起因する動作不良はなかったことから，細切れ時間ワーキングスタイルを活用した開発プロジェクトが遂行可能であることが確認できた．初期参画コストは，通常のプロジェクトより低いと評価され（表 1），おおよそのマイクロタスカーは半日以下で最初のタスクに着手できた．総作業時間が短いエンジニアを活用するうえでのコスト上の課題「タスク着手に求められる初期参画コストが低いこと」が解決されている．「タスク単位が小さいこと」について，マイクロタスカーによりややばらつきはあるが，目安と定めた値を下回ったことが確認できた（図 5）．エンジニアの活動状況から，作業実行時間帯が分散していることが観察され（図 6），アンケートでも，マイクロタスカーらは本業務の合間など 30 分から 1 時間の空きが見込めそうなタイミングで作業を実行したと回答している．つまり，自身の状況のみから作業時間を決定しており，「独立したタスク実行が可能なこと」の達成が確認された．以上により，マイクロタスクプログラミングを採用することで，ソフトウェア開発の実装工程の作業について，「細切れ時間ワーキングスタイル」を実現するうえで必要な 3 つのタスク特性を満たすことが可能であると確認された．

6.2 プロジェクトマネジメント視点の懸念と実態

「エンジニアは小単位でのタスク実行に拒否感を示さないか」に対して，実証実験に参加したエンジニアの 9 名うち，同様のプロジェクトに再度参画したくないと答えたのは 2 名であった（表 2）．アンケートでは，その理由に自身のプロダクトへの寄与が見えにくいことをあげている．2 名以外からも，自身のコードがプロダクトに採用されたかを判断するため，レビュー結果を知りたいというコメントがあった．レビュー結果の通知をはじめ，プロジェクトの貢献を明確に示すことがエンジニアのモチベーションを向上させる可能性がある．

本実証実験では，マイクロタスカーらは副業務として参画しており，業務時間のうちの一部を本プロジェクトに割り当てている．このケースでは，企業内の人材流動性の向上という主に組織の利益が主となる．参加したエンジニアらからも，「タスク実行に必要となる時間が推定されていて，空き時間に合わせたタスクが取得できると複数プロジェクトの掛け持ちが容易になる」との意見が得られている．一方で，長時間・定常勤務が難しい方の活用については，本稿では議論できていない．副業が組織の利点が主であるのに対し，就業機会が得られるという点でマイクロタスカー自身の利点があり，細切れ時間ワーキングスタイルに対する意見も異なることが考えられる，今後，実験・ヒアリングなどによる調査が必要であると考える．細切れ時間ワーキングスタイルは働き方の選択肢の 1 つであり，全員が望ましいと考える働き方である必要はないことに留意する必要がある．

マイクロタスカーからは，マイクロ仕様に不明点があった場合，マイクロタスクに着手したものの完了に至らなかった場合に専任開発者や他の開発者への情報伝達を行いたいという声があった．チャットなど同期性の高いコミュニケーションを求める声もあったが，作業タイミングの自由さ（非同期性）を維持した解決方法を検討することが必要である．エンジニア間のコミュニケーションなど通常の開発プロジェクトと大きく異なる部分があり，適応ユースケースや採用時の考え方のポイントを整理することが，細切れ時間ワーキングスタイルの拡大に必要である．

また，マイクロタスカーの作業タイミングや作業量はコントロールをできないことから，進捗管理の難しさがあげられた．マイクロタスカーの活動状況の把握やそれをふまえた予測，あるいは活動の活性化方法が必要である．

6.3 マイクロタスクへの分割と範囲

プロジェクトマネジメント視点の懸念としてあげられていた「専任開発者の負担が大きくなりすぎないか」に対し，マイクロ仕様の作成は，一般的な開発でのモジュール分割や設計に近く，記述量もほぼ同等であったとのインタビューコメントが得られている．

本実証実験では，マイクロタスクへの分割の指針などは設けておらず，設計者らの判断により，オブジェクト指向プログラミングに基づくモジュール分割が行われ，モジュールの内部機能をマイクロタスクの単位として定められた．これらの判断は，専任開発者のスキルや経験に依存する可能性がある．本実証実験においても初期には，仕様の記述レベルや書き方についての迷いが生じたとのコメントがあった．記述の指針化やコツのノウハウ化がマイクロタスクプログラミングおよび細切れ時間ワーキングスタイル適用拡大に有用であると考えられる．

本実証実験の開発対象は，既存 API を活用したサービスの新規開発であり，開発特性としてシステムが参照するデータ形式やインタフェースの整理が容易であり，モジュール分割やマイクロ仕様の作成を容易にした可能性がある．マイクロタスクプログラミングは，たとえばマイクロサービス [3] のような個々のモジュールの独立性の高いアーキテクチャを採用する案件との親和性が高いと考えられる．一方，大規模システムでは，システム横断的に仕様の把握・調整や，データ構造の詳細の理解が必要なモジュールが存在しうる．この場合には 3.2.1 項で述べた「開発作業の分割」における，ステップ 1 で，専任開発者の担当とし，マイクロタスクに分割できない範囲が多くなることが考えられる．また，既存システムの改修では，システムアーキテクチャ特性により，マイクロタスクへの分割が難しいケースがある．専任開発者の設計とマイクロタスカーによる実装の作業時間の計測・分析を含め，マイクロタスカーに依頼するタスクの選定基準の検討が必要である．

7. 関連研究

7.1 開発タスクのアウトソーシング

ソフトウェア開発プロジェクトにおけるフリーランサーの活用が進んでいる．ソフトウェア開発分野でのフリーランサーの平均勤務時間は週あたり 4.8 日/36 時間であり [16]，長時間労働が問題となっている IT 業界の平均よりは短いもののプロジェクトへの参画の仕方は，フルタイムが前提となっていると見られる．タスクを不特定多数のネットワーク上の個人に対してアウトソーシングするクラウドソーシング [1], [17], [20] の活用も進んでいる．クラウドソーシングが対象とするソフトウェア開発のタスクとして，設計 [13]，実装 [2], [10], [11], [12]，試験 [4] の取り組みが報告されている．不特定多数の技術者が登録するコミュニティを運営し，プロジェクトと技術者のマッチングを支援するプラットフォームも存在し，ソフトウェアのバグの発見など複数名が並行して同じタスクに取り組む [22], [23]．仕様を満たす設計や実装を複数名でチャレンジし，そのうち優れた設計や実装に対して対価が支払われる [21] ことが一般的である．本稿で報告したような，ソフトウェアを実現するための小さな複数のタスクを複数のエンジニアで遂行し，それらの組合せでソフトウェアを作成するという形態での企業での実践事例はない．

7.2 リモートワークでのプロジェクトへの参画コスト

リモートワーク下のプロジェクト参画に関する企業の調査研究では，当該企業に新規雇用された技術者へのインタビューを通じて，プロジェクトの既存メンバとの個人的・物理的なコミュニケーションを行う機会が得られないことが，プロジェクトへの参画の最大の障壁となることが指摘されている [18]．同様に，プロジェクトのすべてのメンバがリモートワークである OSS のプロジェクトでも，メンバの参画コストが高いことが指摘されている [6], [24]．本稿で提案するアプローチでは，参画コスト抑制に向けて，開発タスクを小さく分割することでシステムの全体像を把握せずともタスク実行可能としている．

7.3 ソフトウェアの設計思想（モジュール化）

ソフトウェアは複数のモジュールから構成され，これらが協調動作している．モジュール間には多くの依存関係が存在する．依存関係が複雑な場合には，あるモジュールの変更の影響が依存関係をたどり，広範囲のモジュールに伝搬する．このようなソフトウェアは修正・保守のコストが増大する．モジュール内の凝集度を高め，かつモジュール間の結合度を低くするアプローチが提案されている [5], [14], [15]．適切にモジュール化されたソフトウェアは，各モジュールの責任は明確化され，仕様は独立した振舞いとして定義される．エンジニアは，他のモジュールの中身を過度に意識する必要はなく，他モジュールとのインターフェイスのみ理解すれば，モジュールの実装が可能になる．本稿でも既存のモジュール化のアプローチに基づく設計をベースとしている．

8. まとめと今後の課題

「時間」柔軟な働き方として細切れ時間ワーキングスタイルを提案し，実践上の課題の解決にソフトウェア開発アプローチ「マイクロタスクプログラミング」の採用が有効であると判断した．そこで **RQ：ソフトウェア開発アプローチ「マイクロタスクプログラミング」は，柔軟な働き方の実現に貢献するのか？**の問いを設定し，企業の開発プロジェクトでの実証実験を行った．実証実験では，「マイクロタスクプログラミング」が，細切れ時間ワーキングスタイルが求めるタスク特性を満たし，プロジェクトマネジ

メント視点からの懸念が大きな問題とならなかった．つまり，「マイクロタスクプログラミング」は，ソフトウェア開発の実装工程について柔軟な働き方の実現に貢献するという結論が得られた．細切れ時間ワーキングスタイルの実践を拡大するためには，コミュニケーション方法の改善，マイクロタスカーの活動状況の予測，活動の活性化方法の検討，適応ユースケースの整理が必要である．また，実装工程以外のマイクロタスクによる実行が可能であるかについても今後検討していきたい．

参考文献

[1] Ågerfalk, P.J., Fitzgerald, B. and Stol, K.: *Software Sourcing in the Age of Open*, SpringerBriefs in Computer Science (2015).
[2] Chen, Y., Lee, S.W., Xie, Y., Yang, Y., Lasecki, W.S. and Oney, S.: Codeon: On-Demand Software Development Assistance, *2017 CHI Conference on Human Factors in Computing Systems*, pp.6220–6231, ACM (2017).
[3] Richardson, C.: *Microservices patterns*, Simon and Schuster (2018).
[4] Dwarakanath, A., Chintala, U., Shrikanth, N.C., Virdi, G., Kass, A., Chandran, A., Sengupta, S. and Paul, S.: Crowd build: A methodology for enterprise software development using crowdsourcing, *International Workshop on Crowd sourcing in Software Engineering*, pp.8–14 (2015).
[5] Evans, E.: *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley Professional (2003).
[6] Fagerholm, F., Sanchez Guinea, A., Borenstein, J. and Münch, J.: Onboarding in Open Source Projects, *IEEE Software*, Vol.31, No.6, pp.54–61 (2014).
[7] フレデリック P. ブルックス Jr.（著），山内正弥（訳）：ソフトウェア開発の神話，企画センター (1977).
[8] 飯村結香子，斎藤　忍，際田泰弘，上原潤二：ニューノーマルでの多様で柔軟な働き方を実現するソフトウェア開発アプローチの実践報告，経験報告セッション，ソフトウェアイノベーションシンポジウム (2020).
[9] 経済産業省：平成 30 年度　我が国におけるデータ駆動型社会に係る基盤整備「IT 人材の最新動向と将来推計に関する調査結果」(2016)，入手先 ⟨https://www.meti.go.jp/⟩.
[10] Lasecki, W.S., Kim, J., Rafter, N., Sen, O., Bigham, J.P. and Bernstein, M.S.: Apparition: Crowdsourced User Interfaces that Come to Life as You Sketch Them, *33rd Annual ACM Conference on Human Factors in Computing Systems*, pp.1925–1934 (2015).
[11] LaToza, T., Lecce, A.D., Ricci, F., Towne, W.B. and van der Hoek, A.: Microtask Programming, *IEEE Trans. Software Engineering*, Vol.45, No.11, pp.1106–1124 (2019).
[12] LaToza, T. and van der Hoek, A.: Crowdsourcing in Software Engineering: Models, Motivations, and Challenges, *IEEE Software*, Vol.33, No.1, pp.74–80 (2016).
[13] LaToza, T., Chen, M., Jiang, L., Zhao, M. and van der Hoek, A.: Borrowing from the Crowd: A Study of Recombination in Software Design Competitions, *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (*ICSE*), pp.551–562 (2015).
[14] Meyer, B.: Object-Oriented Software Construction, Prentice Hall (2000).
[15] Myer, G.J.: *Reliable Software Through Composite Design*, Petrocelli Books Inc. (1975).
[16] リクルートワークス研究所：データで見る日本のフリーランス，入手先 ⟨https://www.works-i.com/research/works-report/item/freelance2020_jp_3.pdf⟩（参照 2021-07-31）.
[17] Riehle, D.: The Commercial Open Source Business Model, Nelson, M.L., Shaw, M.J. and Strader, T.J. (Eds.), Value Creation in E-Business Management, *AMCIS 2009*, Lecture Notes in Business Information Processing, Vol.36, Springer, Berlin, Heidelberg (2009).
[18] Rodeghero, P., Zimmermann, T., Houck, B. and Ford, D.: Please Turn Your Cameras on: Remote Onboarding of Software Developers During a Pandemic, *43rd International Conference on Software Engineering: Software Engineering in Practice* (*ICSE-SEIP*), pp.41–50 (2021).
[19] Saito, S., Iimura, Y., Aghayi, E. and LaToza, D.T.: Can Microtask Programming Work in Industry?, *28th ACM ESEC/FSE2020*, pp.1263–1273 (2020).
[20] Stol, K.-J. and Fitzgerald, B.: Two's company, three's a crowd: A case study of crowdsourcing software development, *ACM ICSE2014*, pp.187–198 (2014).
[21] TopCoder (2021), available from ⟨https://www.topcoder.com/⟩.
[22] UserTesting.com (2021), available from ⟨https://www.usertesting.com/⟩.
[23] uTest (2021), available from ⟨https://www.utest.com/⟩.
[24] Von Krogh, G., Spaeth, S. and Lakhani, K.R.: Community, joining, and specialization in open source software innovation: A case study, *Research Policy*, Vol.32, No.7, pp.1217–1241 (2003).

飯村 結香子　（正会員）

日本電信電話株式会社．2001 年熊本大学大学院博士前期課程修了，同年日本電信電話株式会社入社．現在，NTT コンピュータ&データサイエンス研究所に所属．ソフトウェア工学に関する研究開発に従事．

斎藤 忍　（正会員）

日本電信電話株式会社．2001 年慶應義塾大学大学院修士課程修了，同年 NTT データに入社．2015 年日本電信電話株式会社に転籍．2016〜2018 年カリフォルニア大学アーバイン校客員研究員．現在，NTT コンピュータ&データサイエンス研究所に所属．ソフトウェア工学，要求工学に関する研究開発に従事．2007 年慶應義塾大学大学院博士課程修了．博士（工学）．