「時間」柔軟な働き方の実現に向けたソフトウェア開発アプローチの実践と評価

飯村 結香子[1,a)]　斎藤 忍[1,b)]

概要：テレワークの普及は，メンバが一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の現場に，働く「場所」の柔軟性をもたらした．一方，ソフトウェア開発プロジェクトは，全期間を通してフルタイムの参画を求めることが多く，働く「時間」の制約がプロジェクト参画者の条件を狭めている．筆者らはソフトウェアの仕様を小さい部分（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ことにより，ソフトウェアを開発するアプローチ「マイクロタスクプログラミング」に取り組んでいる．本稿では，ソフトウェア開発プロジェクトメンバの「時間」柔軟な働き方の実現に向けたマイクロタスクプログラミングの活用を提示し，企業の開発プロジェクトでの実践評価を報告する．

Practical Report on Software Development Approach for Promoting Workstyle Flexibility

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．国内の生産年齢人口（15 歳〜64 歳）は，1995 年にピーク（約 8,716 万人）を迎えて以降、減少の一途である [10]．IT 業界に視点を絞ると，2015 年時点において既に約 17 万人の IT 人材が不足していたと報告されており，それ以降も IT 人材の供給は減少し続けている [6]．一方，このような状況でも，個人や法人（企業や団体）からの IT へのニーズは増加し続けている．その結果，IT 人材への需要も拡大を続けている．即ち，IT 人材の需給のギャップは広がり続けている．

IT 人材の需給のギャップ（不足）が生み出す問題としては，機会損失と長時間労働の 2 つが挙げられる．どれだけ高い IT のニーズがあっても，その IT を開発するプロジェクトに必要なメンバが確保できなければ，プロジェクトは開始することはできない．プロジェクトが開始できなければ，ビジネスの機会を逃して機会損失が発生してしまう．また，たとえプロジェクトを開始できたとしても，プロジェクト遂行に十分な人材を確保できなければ，プロジェクトのメンバ一人当たりの作業量が増大し，結果として恒常的な長時間労働につながることになる．

雇用動向調査 [7] によると，新たに就業した人の割合（入職率）は，仕事から離れた人の割合（離職率）を 7 年連続で上回っている．即ち，国内の概況は「入職超過」の状況が続いている．それにも関わらず，IT 業界では人材不足の状況が解消されていない．筆者らは，IT 業界が現在の状況に至った理由の 1 つは，ソフトウェア開発プロジェクトにおける働き方の柔軟性の低さにあると考える．従来のソフトウェア開発では，プロジェクトの全期間を通してフルタイムで従事するスタイルをメンバに求めるのが一般的である．一方で，このような働き方のスタイルでは，働く「時間」に制約がある人たちではプロジェクトに参画することができない．例えば，介護者や育児者の方たちは，フルタイムで働くのが難しい，あるいは，勤務予定の変更が起きやすいなどの制約がある．

これからのソフトウェア開発は，プロジェクトへの参画期間，作業時間帯の自由度が高い，柔軟な働き方のスタイルを実現することが重要と考える．現在のソフトウェア開発のプロジェクトでは活かし切れていない IT 人材（働く時間に制約がある人たち）の活用につながるからである．更に，企業内の IT 人材の流動性向上 [3], [5] にもつながる．

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

© 2021 Information Processing Society of Japan

例えば，プロジェクトの狭間の限られた期間のみ参画する，といった参画も可能となる．

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」[9] では，ソフトウェアの仕様を小さく分割し，複数人に依頼する．マイクロタスクプログラミングのタスクは短時間で独立に実施可能であることから，「時間」に柔軟な働き方の実現に貢献すると考えられる．そこで本稿では，はじめに柔軟な働き方「細切れ時間ワーキングスタイル」を定義する．その上で企業での実践事例 [4] をもとに細切れ時間ワーキングスタイルの要件を掘り下げ，要件充足に向けたマイクロタスクプログラミングの技術的な特徴を提示し，実践による評価・考察を行なう．

本稿の構成は以下のとおりである．2 章で，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，ソフトウェア開発プロジェクトで実現するための課題を示す．3 章ではマイクロタスクプログラミングを概説する．細切れ時間ワーキングスタイルの実現に向けた，マイクロタスクプログラミングの技術的な特徴を提示する．4 章では，企業のソフトウェア開発プロジェクトにおけるマイクロタスクプログラミングの実践事例を示す．事例の内容に基づき，細切れ時間ワーキングスタイルの実現可能性を 5 章で評価する．6 章で考察し，最後の 7 章で結論を述べる．

2. 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

1 章で述べたソフトウェア開発プロジェクトに参画する上での「時間」の制約には 2 種類があると考える．本稿では「時間」柔軟な働き方として，この 2 種類の「時間」の制約を排除することを目指す．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，例えば 6 ヶ月のプロジェクト期間の途中からの参画，および離脱など部分的な参画を許容する．2）では定常的かつフルタイムの参画を求めず，非定常的に短時間に作業を行うことを許容する．また，1），2）いずれについても，事前に確定できないことを前提とする．これらより本稿では，「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できる働き方と定義する．決定は事前ではなく，適宜変更可能とする．その上で，「細切れ時間ワーキングスタイル」のメンバと定常従事者（以降，専任開発者と呼ぶ）との組合せによりプロジェクト遂行を目指す．細切れ時間（Fragmented-time）は，全時間（Full-time）と対極の言葉として位置付けている．

2.2 「細切れ時間ワーキングスタイル」の実現への課題

ソフトウェア開発プロジェクトを，定常従事者と，細切れ時間ワーキングスタイルを選択するエンジニアで構成し，遂行するための課題を明らかにする．「細切れ時間ワーキングスタイル」でソフトウェア開発に参画するために，実施する作業（タスク）が持つべき特性と，「細切れ時間ワーキングスタイル」でのプロジェクト参画に対し，現場の方がどのような懸念を持つかについて整理する．

2.2.1 「細切れ時間ワーキングスタイル」で実施可能なタスク特性

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できるために，実施するタスクがどのような特性を持つ必要があるかを整理する．

- **課題 A1「タスク着手までの初期参画コストが低いこと」:**通常の開発では，タスクの実施にスキルや経験を備えるだけでなく，プロジェクト知識が求められる．このため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加した人は, 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.”と言われる [1]．プロジェクト知識の習得にかかる時間を初期参画コストと呼ぶ．長期間，定常的にプロジェクトに参画する場合は，初期参画コストは総作業時間に比して小さくなる．しかし，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，初期参画コストを低く抑える必要がある．
- **課題 A2「タスクが短時間で完了可能なこと」:** 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．着手したタスクがエンジニアが希望する時間長で終わらない可能性を抑えるため，タスクは短時間で完了できなければならない．後述する実践事例では，エンジニアが実施するタスク単位は 1.5 時間程度の作業で完了できることを目安とした．ただし，1 回の作業で 1 つのタスクが完了できない場合には，中断を挟み複数回の作業での実施も許容する．タスク単位が小さいことは，何らかの事情によりタスクの完遂ができなかった場合のリスク低減にもつながる．
- **課題 A3「タスクが独立に遂行が可能なこと」:** 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れない．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にできないことを意味する．タスクの割り当てのために打ち合わせを行なったり，タスクを実施する上で他のエンジニアと協調作業を行うことはできない．エンジニアが希望する任意のタイミングで，その時に実施可能なタスクがエンジニアに割り当てられ，タスクの遂行に際しては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

細切れ時間ワーキングスタイルについて，ソフトウェア

© 2021 Information Processing Society of Japan

開発の現場の実務者と意見交換をしたところ，プロジェクトマネジメント視点から複数の懸念が挙げられた．実践を通じて得られた客観的な事実に基づき，これらの懸念を払拭することが必要といえる．

- **懸念 B1 「細切れ時間ワーキングスタイルをエンジニアは受容するか」：**従来の開発では作業単位が大きく，また，仕様を理解するために関連ドキュメントを読み込むことはむしろ推奨されている．細切れ時間ワーキングスタイルの，小さく部分的な作業というプロジェクトとの関わり方に対して，エンジニアは否定的な態度を示すのではないかと懸念された．
- **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エンジニアは，プロジェクト知識を持たず，同期的なコミュニケーションも取れないことから，様々な作業のしわ寄せが専任開発者にくることになる．加えて，細切れ時間ワーキングスタイルのメンバが，タスクを短時間で完了できるように，専任開発者は通常よりも分割・詳細化された仕様書を作成するするな仕様を作成する必要がある．専任開発者は，作業量が増えること，従来とは著しく異なる作業が求められることが懸念された．
- **懸念 B3 「成果物が特定の作業者に依存しないか」:**通常のプロジェクトであれば，エンジニアはモジュール等を単位として作業を行う．テストで発見された成果物の不具合修正，仕様書の変更や改修も，作成したエンジニアが担当することが多い，これは，作成者以外が成果物を理解するのが困難だという考えに基づく．しかし，細切れ時間ワーキングスタイルでは，プロジェクト期間中に，エンジニアがプロジェクトを離れる可能性が高い．また，プロジェクトに参画中であっても，そのエンジニアが依頼時に作業実施可能であるとは限らない．このため，作成したエンジニアにしか成果物が理解できず，改修や保守がしづらい状態になることが懸念された．

3. マイクロタスクプログラミング

マイクロタスクプログラミングは，ソフトウェアの仕様を小さい部分（マイクロ仕様）に分割し，それぞれのマイクロ仕様を実現するタスク（マイクロタスク）を複数人が独立に実行することにより，ソフトウェアを開発するアプローチである [9]．本章では，本アプローチの主要な構成要素（2 つの役割とマイクロタスク）と技術的な特徴を解説する．併せて，細切れ時間ワーキングスタイルを活用する上での課題がマイクロタスクプログラミングにより解決すると考えられる理由を示す．

3.1 主要な構成要素

本アプローチの主要な構成予想である 2 つの役割とマイクロタスクの関係を**図 1** に示す．本図において，役割の横に位置する矩形の横幅は，作業に従事する期間を表す．

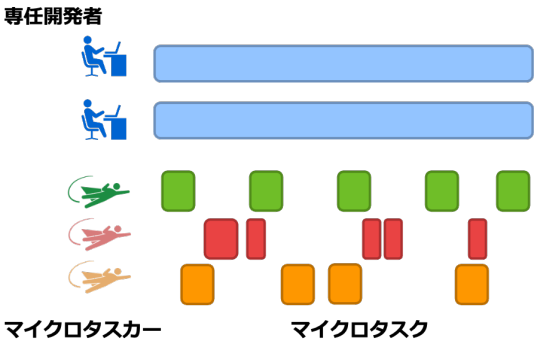

図 1 マイクロタスクプログラミングの主要な構成要素
Fig. 1 Main technical components in microtask programming.

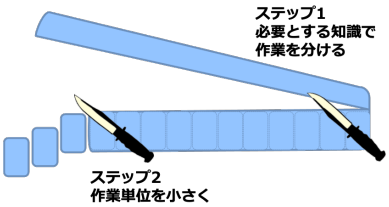

図 2 ソフトウェア開発作業の分割のイメージ
Fig. 2 Software task decomposition.

- **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．図 1 における専任開発者の横に位置する細長い矩形は，プロジェクト全期間，定常的に作業していることを表す．
- **マイクロタスカー：**プロジェクトへの参画を宣言し，各自の任意のタイミングで断続的に短時間の作業を実施する役割．図 1 における専任開発者の横に位置する横幅がそれぞれ異なる矩形は，各自が細切れ時間で作業をしていることを表す．細切れ時間ワーキングスタイルを選択するエンジニアは，マイクロタスカーとしてプロジェクトに参加する．
- **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，独立で完結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」

ソフトウェア開発の実装工程の作業を 2 段階に分割をする．**図 2** に作業の分割のイメージを示す．

- **ステップ 1.** ソフトウェア開発作業を，業務やシステム全体の知識が必要な部分と，不要な部分に分割する．前者は専任開発者が担当し，後者をマイクロタスカーが担当する．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であれば，当該モジュールの仕様のみを理解すれば作業可能であ

© 2021 Information Processing Society of Japan

る．これによりマイクロタスカーに割り当てられるタスクは，プロジェクト知識の事前習得が不要であり，課題 A1「タスク着手までの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.** 前ステップで分割された，業務やシステム全体の知識が不要な部分に対して，他の作業に依存しない，小さな単位に分割を行う．分割後の小さな単位の作業はマイクロタスクと呼ばれる．

3.2.2 特徴 2「仕様の分割と割当て指針」

ソフトウェアは複数のモジュールに分割して構成される．開発者はモジュールを単位として作業を行うのが一般的である．マイクロタスクプログラミングでは，モジュールをさらに複数の小さな部分（マイクロ仕様）に分割する．1 つのマイクロ仕様は，実装が完了したかが，単体テストで動作確認可能な小さなステップ単位とする．マイクロ仕様はモジュールごとに記述する（**表 1** 参照）．モジュールの分割を設計した時点で，モジュール名，モジュールの概要，モジュールの入出力データ構造を記載する．データ構造には，モジュール間共通で使用するために定めたユーザ定義型を利用する場合もある．モジュール概要を実現するための，小さな部分をマイクロ仕様として簡略に記述する．例では，番号付き箇条書きの 2 つの項目が一つのマイクロ仕様である．そして，1 つのマイクロ仕様を実現するために行われる作業をマイクロタスクと呼ぶ．一つのモジュール仕様をさらに分割したマイクロ仕様は，その実現にかかる時間は小さく，課題 A2「タスク単位が小さいこと」が実現できると考えられる．

マイクロ仕様の実現のためには，実装（ソースコードとテストコードの作成）とレビューの 2 種類のマイクロタスクが実施される．また，同一モジュール内のマイクロタスクは直列に実施する．あるマイクロ仕様の実装とレビューのマイクロタスクは，異なるメンバで実施するように割り当てを行う．**図 3** にマイクロタスクの割り当てイメージを示す．図の例では，1 つのモジュール仕様が 2 つのマイクロ仕様分割されている．各マイクロ仕様に対して，実装とレビュー 2 種のマイクロタスクが行われるため，計 4 つのマイクロタスクが直列に実施され，モジュール A が完成する．この時，マイクロ仕様 1 の実装とレビューのマイクロタスクは異なる（色が異なる）2 人のマイクロタスカーに割り当てられている．

表 1 マイクロ仕様の例．[9] の Figure4. を著者らが日本語訳
Table 1 Example of Microspecifications.

モジュール名	チェックシート更新
モジュール概要	入力データによりチェックシートデータを更新する
入力データ型	checksheetData
返却データ型	文字列型
マイクロ仕様	1. 入力パラメータ checksheetData から DBChecksheetUpdateInfo 型の配列を作成し、3rd party API の updateObjectImplementation に渡す - （パラメータの変換指定） 2. パラメータを以下の条件でチェックし、エラーがある場合、Error をスローする． - メッセージは「パラメータが不正です。」 - （バリデーションの条件）

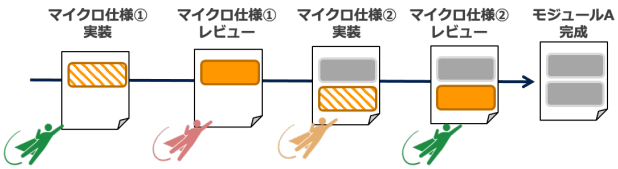

図 3 マイクロタスクの割り当てのイメージ
Fig. 3 Microtask programming workflow.

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 4** 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図 4 の各フローにおける実施内容を以下に記す．

(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記述する．ソースコードテンプレート他の開発資材と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，割り当て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，割当てられ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．

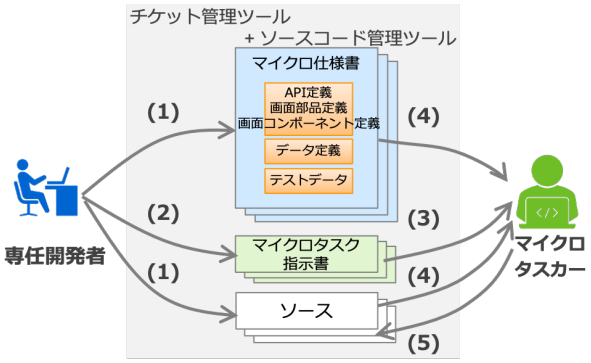

図 4 役割間のモノ・情報の流れ
Fig. 4 Communication flow.

© 2021 Information Processing Society of Japan

表 2 課題と技術的な特徴の対応表
Table 2 Issues and technical features table.

	特徴 1. 開発作業の分割	特徴 2. 仕様の分割と割当て指針	特徴 3. 非同期コミュニケーション
課題 A1. タスク着手までの初期参画コストが低いこと	X (Step1)		
課題 A2. タスクが短時間で完了可能なこと		X	
課題 A3. タスクが独立に遂行可能なこと			X

(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．

このフローの作業実施では，他のメンバとのコミュニケーションや協調を必要とせず，つまり，課題 A3「独立にタスク遂行が可能なこと」を実現できると考えられる．

マイクロタスカーは専用の開発環境で作業を実施する．開発環境は，タスク割り当てや作業結果のコミット等のイベントに応じて，チケット管理ツールのチケット更新等を自動で行う．チケット管理ツールから，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抽出できる．

3.4 課題と技術的な特徴の対応

前章で挙げた「細切れ時間ワーキングスタイル」で実施可能なタスクの特性と，「マイクロタスクプログラミング」の技術的な特徴の対応関係を**表 2** に記す．「細切れ時間ワーキングスタイル」で実施可能なタスクの 3 つの特性は，マイクロタスクプログラミングを開発アプローチの技術的な特徴に実現されると考えた．一方，プロジェクトマネジメント視点の課題は，マイクロタスクプログラミングの技術的特徴から達成されるかは議論することはできないため，実践で得られた実態から議論する必要であると考えた．以上について現場での実践を通して，検証することとした．

4. 実践

4.1 実践に至る背景

実践した企業では，社員のライフイベント（育児，介護）やワークスタイルの多様化（リモートワーク，短時間・分断勤務）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では，今後ますます社員の参加が難しくなるとの認識があった．このため，細切れ時間ワーキングスタイルをソフトウェア開発プロジェクトで活用できるのであれば，時間制約のある社員であっても，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された．また，シニア技術者にとっても，

「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．
社内プロダクト API を利用します．関連ドキュメントは事前に参照ください．開発プロダクトの仕様の読込みなどは不要です．
必要スキル：REST API クライアント開発，JavaScript (CoffeeScript)，HTML5/Web アプリの基礎知識

図 5 参加者募集の案内文
Fig. 5 Participant recruitment for case study.

細切れ時間を活用することで，新たなプログラミング言語等の知識やスキルを身につけながら，プロジェクトにも参画できる．そのため，実践的なリカレント教育の効果も期待された．そこで，前章のマイクロタスクプログラミングを実プロジェクトで実践し，細切れ時間ワーキングスタイルの実現可能性を検証することとした．

4.2 開発プロジェクトの体制

本プロジェクトは，専任開発者 3 名，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ーは，社内で実証実験の趣旨および参加条件を公開し，希望者を募った (**図 5** 参照)．

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エンジニアには，育児，介護や自身の療養などにより，専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ある．本プロジェクトで細切れ時間ワーキングスタイルを選択するエンジニアは，後者である．彼らの本プロジェクトの作業は，勤務日の 9 時〜19 時の範囲で行うものとした（残業時間中での細切れ時間の活用は原則禁止）．また，参加期間は 3 週間程度，週あたりの参加時間は 6 時間程度を見込めることを作業の目安とした．ただし，実際の作業時間数は各エンジニアに委ねられた．

本実証実験の専任開発者（定常従事）とマイクロタスカー（細切れ時間ワーキングスタイルを選択したエンジニア）は，同じ会社に属している．しかしながら，共通のプロジェクトに従事した経験等はなかった．本プロジェクトの開始以前に，顔合わせなどの関係性の構築は行わなかった．

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 PoC であった．プロダクトは大きく 3

© 2021 Information Processing Society of Japan

つの機能をもち，16 クラスから構成された．本プロジェクトでは，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 つから 3 つのメソッドを備えており，全体で 32 メソッド，即ち，32 マイクロ仕様が準備された．一つのマイクロ仕様に対し，実装とレビュー 2 種の作業を実行するため，マイクロタスク数は 64 であった．開発言語は Type Script が使用された．

4.3.1 開発プロジェクトの経過

全体スケジュールを**図 6** に示す．実現機能は，専任開発者を担当したエンジニアらが以前より検討しており，本実証実験に当たって仕様確定，マイクロ仕様設計を行なった．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タスクプログラミングは（マイクロ仕様の実装& UT/レビュー）約 1 ヶ月で行われた．同期間に専任開発者は，マイクロタスカーからの問い合わせ対応，またマージ作業を行なった．全てのマイクロタスクが完了した後に結合試験を行なった．

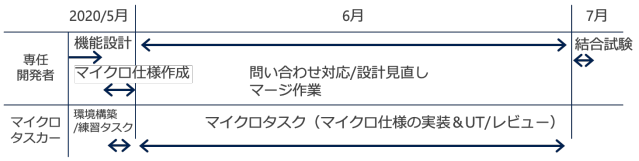

図 6 プロジェクトスケジュール
Fig. 6 Actual development schedule.

期間中のマイクロ仕様の公開数，完了数の遷移を**図 7** に示す．マイクロ仕様の公開数が折れ線で，完了数が棒グラフで示されている．灰色がけは週末（作業実施不可日）を示している．一つのマイクロ仕様に対し，実装が行われ，別のエンジニアによりレビューされる．レビューで実装に問題がないと判定されれば，そのマイクロ仕様は完了する．

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方向へとのぼる階段状となっている．第 2 週の後半（▼マークの箇所）にモジュール間で仕様の不整合が判明し再設計を行なったため，公開数が減少している．

専任開発者およびマイクロタスカーの作業の結果，予定機能は全て実現された．コードの約 7 割（LoC ベース）はマイクロタスカーにより実装された．専任開発者によりマージ/テスト時に 2 つの動作不良が発見されたが，専任開発者の設定ファイルの記述ミスおよび動作仕様の曖昧さによるもので，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固有の問題ではない．

5. 評価

マイクロタスクプログラミングの採用により「細切れ時間ワーキングスタイル」で実施可能なタスクの特性，課題 A1 から A3 が達成されたか，また，プロジェクトマネジメント視点の懸念の実態がどうであったかを述べる．

5.1 課題 A1「タスク着手までの初期参画コストが低いこと」の達成状況

マイクロタスカーとして参画したエンジニアらに，本プロジェクトへの参画の時間・手間が，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？をとても小さいからとても大きいの 5 段階で尋ねた結果を**表 3** に示す．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．

実際にかかった時間は，30 分から数時間との回答が多かった．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であると答えている．最長は，1 日半であり，今回と普段の開発環境で設定の衝突が起き，この解消のために時間がかかっていた．参画に要するコストが従来の開発プロジェクト低い要因としては，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挙げている．プロジェクトでの最初の作業に着手するまでの時間が数時間であることから，「タスク着手までの初期参画コストが低いこと」が達成されたと言える．

表 3 アンケート結果（初期参画コスト）
Table 3 Questionnaire result about onboarding.

質問：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？				
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

5.2 課題 A2「タスクが短時間で完了可能なこと」の達成状況

タスクが，短時間で完了可能であったかを，マイクロタスカーの作業時間より評価する．本実践では 1 タスクが 1.5 時間での完了を目安とした．マイクロタスカーごとの総作業時間と実施作業数を**図 8** に示す．丸の位置が実装& UT とレビューそれぞれの実施作業数を示し，棒グラフが実装& UT とレビューそれぞれの総合作業時間を示している．1 作業数あたりの作業時間は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．以上から，「タスクが短時間で完了可能なこと」は達成されたと言える．

5.3 課題 A3「タスクが独立に実行可能なこと」の達成状況

本プロジェクトでは，マイクロタスカーに対し，定例・非定例の打ち合わせへの参加，コンタクト可能時間帯の設定を求めていない．

© 2021 Information Processing Society of Japan

図 7　マイクロ仕様の公開数と完了数の遷移

Fig. 7　Burn up chart showing the number of transition of micro specifications assinged to microtask workers and the number of microspecificatoins developped by them.

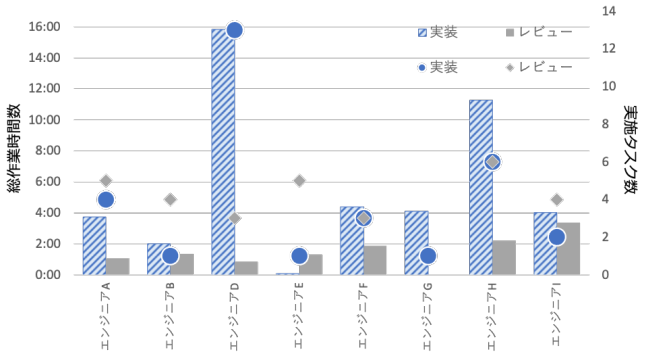

図 8　エンジニア毎の総作業時間，マイクロタスク数

Fig. 8　Total working time and Number of microtasks.

事後アンケートで，どのような時に作業を行なっ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施した」，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施した」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により示す**図 9**．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．矩形色が作業を行な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，独立してタスクが遂行されたことがわかる．また，マイクロタスカーごとにも一定のパターンを見出すことはできず，アンケートの「本業の合間」や「作業が実施可能なタイミング」でタスクを実施したことが裏付けられる．

以上から，筆者らの想定する細切れ時間ワーキングスタイルに近い形で作業が行われており，「タスクが独立に実行可能なこと」が達成されたと言える．

表 4　アンケート結果 (次への参加意欲)

Table 4　Questionnaire result about willingness.

同じやり方のプロジェクトへ次回以降も参画したいか？				
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

5.4　懸念 B1 「エンジニアは細切れ時間ワーキングスタイルを受容するか」の実態

実践前には，概要情報等を与えられずマイクロ仕様のみで作業をすることに不満を抱くのではないか？という懸念が強く持たれていた．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 4** 参照）エンジニアらの受容が低かったとは言えない．

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

実践前には，専任開発者の作業内容や量が，一般的なソフトウェア開発と比較して大きく異なることが懸念された．しかし，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

© 2021 Information Processing Society of Japan

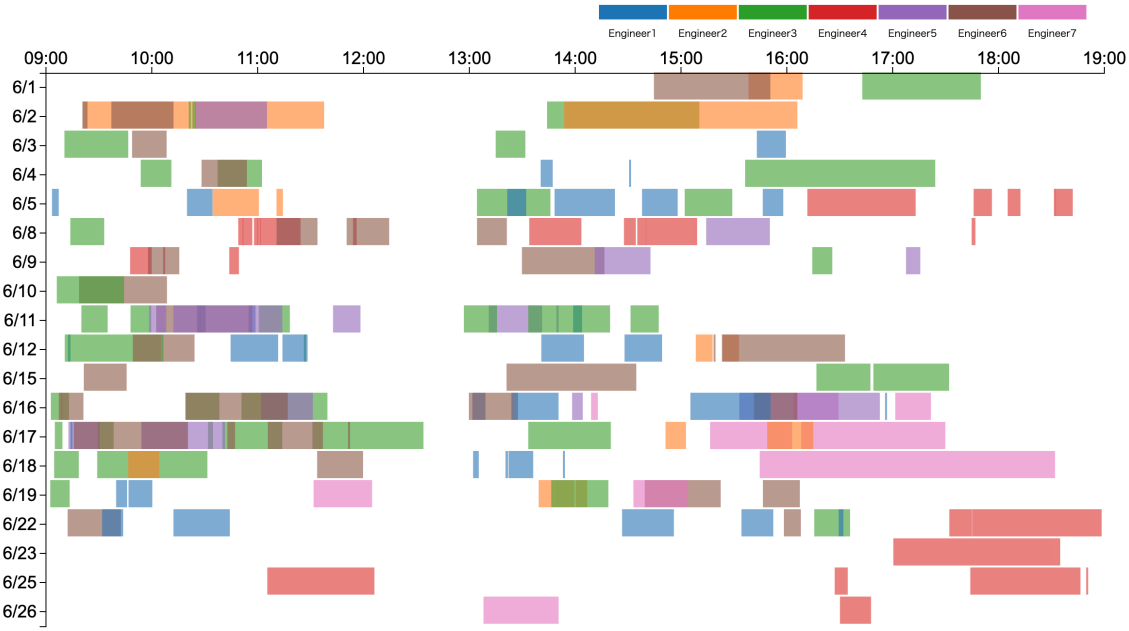

図 9 マイクロタスカーの作業時間帯
Fig. 9 microtask workers' workingtime.

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．モジュール内の分割については，関数単位とすると決めたことで作業中に悩むことはなかった」としている．一般的な開発と比して，専任開発者の作業内容や負担が大きく変わるわけではないと推測される．

5.6 懸念 3「成果物が特定のエンジニアに依存しないか」の実態

実践前には，成果物の内容が特定のエンジニアにしか理解できない状態で，改修や保守がしづらい状態にあることが懸念された．マイクロタスクプログラミングでは，一つのモジュールに関わる作業を複数名が関わり，相互にレビューを行うよう作業フローが設計されている．このため，おのずからその内容を複数名が理解しており，また理解できる状態にあると考えられる．

実態がどうであったかを確認するため．クラス毎のマイクロタスカーの貢献を**図 10** に示す．一つの矩形がそのクラスに関わるマイクロタスク（実装& UT/レビュー）を表し，矩形の色が担当したエンジニアを示す．全てのクラスは 2 名以上のマイクロタスカーに実現されている．以上より，「成果物が特定のエンジニアに依存しないか」の懸念は低いと考えられる．

6. 考察

6.1 マイクロタスキングによる「細切れ時間ワーキングスタイル」の実現可能性

評価対象の開発プロジェクトでは予定機能が全て実現さ

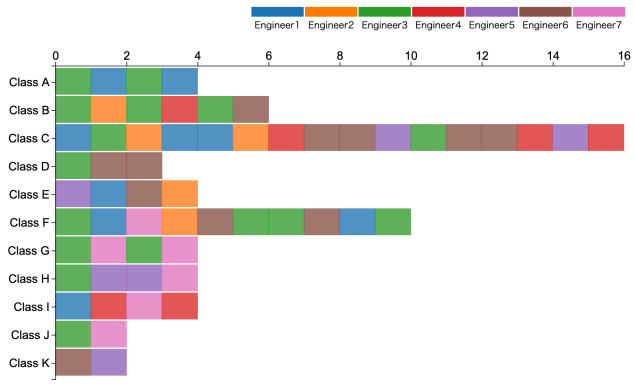

図 10 マイクロタスカーの貢献
Fig. 10 microtask workers' contributions.

れた．また，開発プロダクト，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．これにより，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，30 分から数時間程度であったことから，総作業時間が短いエンジニアを活用する上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，エンジニアによりややばらつきは見られるが，時間のかかる実装作業でも目安を下回ったことが確認できた．エンジニアの活動状況から，作業実施時間帯が分散していることが見て取れ，アンケートでも，エンジニア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施した，つまり，自身の状況のみから作業時間を決定回答

© 2021 Information Processing Society of Japan

しており，「独立にタスク遂行が可能なこと」の達成が確認された．以上より，マイクロタスクプログラミングを採用することで，細切れ時間ワーキングスタイルが求める作業特性を満たすことが可能であることが確認された．

6.2 プロジェクトマネジメント視点での懸念と実態

「エンジニアは細切れ時間ワーキングスタイルを受容するか」について，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．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含め，プロダクトへの貢献をように示すことでエンジニアの受容性をより向上させる可能性がある．その他，マイクロタスカ―からの指摘事項として，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などの情報伝達手段の改善が挙げられた．開発環境等のさらなる利便性の向上は，エンジニアの細切れ時間ワーキングスタイルの受容性に寄与する可能性がある．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．また，細切れ時間ワーキングスタイルは働き方の選択肢の一つであり，必ずしも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．

「専任開発者の負担が大きくなりすぎないか」について，マイクロタスカー向けに行う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いう実態がインタビューよりわかった．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り，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．プロジェクトの途中，モジュール間で仕様の不整合が判明したことから，一部タスクの取り下げが行われた．モジュール間での仕様不正後が，手戻りの要因となりうることは一般的な開発と同じであるが，「マイクロタスクプログラミング」開発アプローチでは，ミーティング等でのコミュニケーションを行わないため，タスクの依存関係，またそれに基づくタスクの公開・非公開管理支援システムなどの検討が今後必要である．ごく小さいモジュールは，専任開発者が作成した方がコストが低いのではという意見もあり，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. おわりに

本稿では，ソフトウェア開発のプロジェクトの現場に，働く「場所」の柔軟性に加え「時間」の柔軟性をもたらす，細切れ時間ワーキングスタイルの活用を提案した．細切れ時間ワーキングスタイルをソフトウェア開発プロジェクトで活用する上での課題および懸念を整理し，その課題の解決に対し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．企業の開発プロジェクトへの適用した結果，プロジェクトの遂行が可能であること，「タスク着手までの初期参画コストが低いこと」，「タスクが短時間で完了可能なこと」，「タスクが独立に実行可能なこと」の 3 つの課題が達成されたことが確認され，「マイクロタスクプログラミング」採用による実現可能性が示された．さらに，現場が持つプロジェクトマネジメント上の懸念についても実態が示された．今後，細切れ時間ワーキングスタイルの受容性をさらに向上させるにはマイクロタスカーのコミュニケーション方法の改善，活動状況の把握・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．

参考文献

[1] F. P., Addison Wesley:『ソフトウェア開発の神話』, 企画センター,1977
[2] Ågerfalk, P. J., Fitzgerald, B., Stol, K: Software Sourcing in the Age of Open. SpringerBriefs in Computer Science (2015).
[3] Gurbani, V., Garvert, A., Herbsleb, J.D.: A case study of a corporate open source development model. in: 28th international conference on Software engineering (ICSE '06), May 2006, pp. 472-481.
[4] 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.
[5] InnerSource Commons(online), 入手先 ⟨https://innersourcecommons.org/⟩ (2021.05.01).
[6] 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」, 入手先 ⟨https://www.meti.go.jp/⟩（2016）
[7] 厚生労働省：令和元年 雇用動向調査, 入手先 ⟨https://www.mhlw.go.jp/⟩（2020）.
[8] Riehle D.: The Commercial Open Source Business Model. In: Nelson M.L., Shaw M.J., Strader T.J. (eds) Value Creation in E-Business Management. AMCIS 2009. Lecture Notes in Business Information Processing, vol 36. Springer, Berlin, Heidelberg.
[9] Saito, S., Iimura, Y., Aghayi, E., LaToza, D. T.: Can Microtask Programming Work in Industry?. In: 28th ACM ESEC/FSE2020, pp. 1263–1273.
[10] 総務省：情報通信白書，平成 28 年度版，入手先 ⟨https://www.soumu.go.jp/⟩(2016).
[11] Stol, K.-J., Fitzgerald, B.: Two's company, three's a crowd: a case study of crowdsourcing software development. In: ACM ICSE2014, pp. 187-198.

© 2021 Information Processing Society of Japan