グローバルに分散したチームによるアジャイル開発の課題とその解決策

熊谷 賢　堀内 芳雄

How to resolve communication issues of agile software development in globally distributed teams

Ken Kumagai and Yoshio Horiuchi

複数のコンポーネントから構成されるミドルウェアを複数の国に分散した数十人規模の開発チームによりアジャイル・プロセスで開発する際，複数のタイムゾーンが存在することによって生じるさまざまなコミュニケーション上の課題に直面する．それらの課題に対し，本稿ではコラボレーション・ツールを介した非同期・自発的なコミュニケーションを促進するアプローチをとることで，地理的・時間的に分散した開発メンバー同士が効率的・効果的にコミュニケーションをとることを可能にし，複数の国に分散した開発チームでアジャイル・プロセスがうまく機能することを確認した．

A globally distributed team faces problems of communication to develop middleware that is composed of several components in agile process because members in the team belong to different time zones. This paper proposes an approach to communicate asynchronously and voluntarily by using a collaboration tool. By this approach, this paper confirms that the globally distributed team can develop the middleware in agile process.

Key Words & Phrases：分散アジャイル開発，スクラム，ラショナル・チームコンサート
distributed agile development, Scrum, Rational Team Concert

1. はじめに

ソフトウェア開発にアジャイル・プロセスが適用される事例が増えてきている［1］．アジャイル・プロセスは，従来の前工程が完了しないと次工程に進まないウォーターフォール・モデル［2］と異なり，開発対象を多数の小さな機能へ分割し，1つの反復で1つの機能の開発を目指し，反復の繰り返しにより，機能を追加開発していく手法である．アジャイル・プロセスは，単一の開発手法を指す語ではなく，迅速かつ適応的にソフトウェア開発を行う開発手法群を総称する語であり，一例ではスクラム［3］が挙げられる．スクラムの場合，スプリントと呼ばれる1～4週間の反復を定義し，バックログと呼ばれる製品に必要な項目の一覧の中から，項目を取り出し開発者に割り当てる．チーム・メンバーの中から，スクラム・マスターを選出し，スクラム・マスターはデイリー・スクラムの開催を促す．デイリー・スクラムは，スプリント期間中毎日開催され，進捗を確認する．スプリントの終了時には，スプリント・レビューを行う．

提出日：2012年8月24日

アジャイル・プロセスは，一箇所に集まった小規模なチームに有効であるといわれており，グローバルに分散したチームにおいてアジャイル・プロセスによる開発を行う場合，時間的・距離的な制約が発生し，従来のアジャイル・プロセスで宣言された原則［4］［5］のみでは対応しきれない課題が出てくる．例えば，チームのメンバーがグローバルに分散している場合，会議室で互いに向かい合って話すことは現実的に困難である．筆者らは，グローバルに分散したチームによるアジャイル開発のプロジェクト（以下，プロジェクトA）に携わっており，さまざまな課題に遭遇している．本稿では，そのプロジェクトで実際に発生した課題を説明し，それらに対する解決策を示す．

2章では，関連研究に絡めて本稿の立場を説明し，3章では筆者らが担当するグローバルなプロジェクト・チームとその開発形態について述べる．4章と5章では，その開発における課題と解決策を示し，6章でまとめと今後の展望について述べる．

2. アジャイル・プロセスの関連研究

開発チームは，**図1**に示すようにチーム内メンバー

地理軸＼時間軸	同一タイムゾーン	異なるタイムゾーン（時差の大きな場合）
同一地点	従来の開発	
分散	国内、もしくは、近隣国へのアウトソーシング	本稿の対象

図1. チーム・メンバーの地理的・時間的な分散による分類

が地理的，時間的に分散しているか否か，すなわち，地理軸（同一地点，分散），時間軸（同一のタイムゾーン，異なるタイムゾーン）により4領域に分割することができる．これまでにアジャイル・プロセスを適用した開発が一定の成果を上げたと報告されているが，これは開発チームのメンバーが同一地点に集まって行う開発形態であり，**図1**左上の領域に相当する．

レフィングウェル［6］は，アジャイル・プロセスを企業レベルにスケールアップする際の課題およびプラクティスについて説明している．アジャイル開発においてスケールアップを想定すると，すべての開発は地理的分散開発になると指摘している．大規模開発においては，コンポーネントにあわせてチームを組織化することを述べており，各チームにスクラムを運用し，チーム横断でのコミュニケーションを行うためにスクラム・マスター同士が一堂に会してスクラムを行うスクラム・オブ・スクラムについて述べている．レフィングウェルのアプローチは，大規模開発においてアジャイル・プロセスを適用する際に，チームに属するメンバー数についての指針を与えてくれるが，その結果，出来上がるチームに属するメンバーが地理的，時間的に分散してしまうような場合については言及されていない．

プロジェクトAの場合も，チームはコンポーネントごとに存在するが，ビジネス上の要件にも影響を受ける．例えば，プロジェクトAの場合，お客様がさまざまな国に存在しているため，1つのチームのメンバーが日本以外に，米国，中国，インドなど複数の国に点在している．このような場合，1つのチーム内に複数のタイムゾーンが存在し，かつ，相対的に時差が大きくなる．本稿では，1つのスクラム・チームのメンバーが複数のタイムゾーンで共同開発する場合を対象としており，これは**図1**の右下の領域に相当する．以降，本稿で分散アジャイル開発という語は，**図1**右下の領域の開発形態を指す．

図1左下の領域に相当する開発形態としては，国内，または，近隣国へのアウトソーシングが該当する．チーム内メンバーが地理的に分散することになるが，メンバーの時間的な分散度合いは相対的に小さい場合であり，本稿では対象外としている．また，アウトソーシングの場合，チーム内メンバーの地理的，時間的な分散以外に，通常，発注元と発注先の会社は別会社であるため，開発形態に契約上の制約が発生し，コミュニケーションの在り方にも影響を及ぼす．一方，本稿では主にグローバルに展開する1つの企業体が各国に点在する開発チームにおいて開発を行う場合を想定しており，チーム・メンバーは地理的，時間的に分散しているという状況以外からは制約を受けないものとする．

3. グローバル開発チームのスクラム運用形態

複数のコンポーネントからなるプロジェクトAは，約50人で開発しており，一般的な製品開発形態と同様にコンポーネントごとに約10人のチームが存在し，チームごとに存在するスクラム・マスターがチームの運営のサポートを行う．チームは，デイリー・スクラムを開催し，各メンバーは，それぞれ三つの項目，①前回のデイリー・スクラムから行ったこと，②次回のデイリー・スクラムまで行うこと，③問題点の有無，について報告を行う．デイリー・スクラムで問題があることを確認すれば，会議中または後に，当事者同士で議論をして解決を図る．デイリー・スクラムは，チームの作業進捗を評価するとともにコミュニケーションの改善を行う．また，スクラム・マスター同士が集まり会議を行うスクラム・オブ・スクラムが開催される．プロジェクトAのコンポーネントは，相互に関係しているので，他のチーム・メンバーとやり取りをすることもあるが，スクラム・オブ・スクラムはこのようなチームを横断してチーム・メンバー同士がコミュニケーション行うことをサポートする．

筆者らが担当するコンポーネントBは，プロジェクトAに含まれる複数のコンポーネントのうちの1つであり，

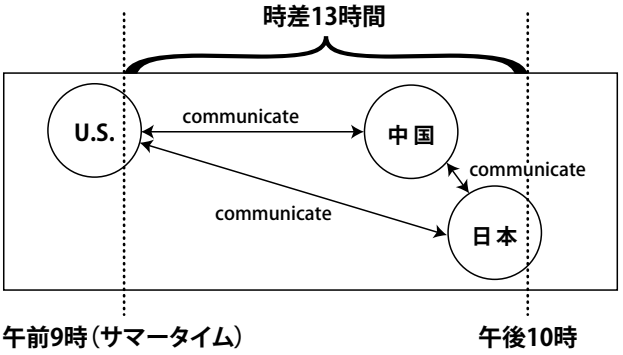

図2. コンポーネントB開発チームの所在

開発チームは，約 15 人で，米国，中国，日本にチーム･メンバーが存在しており，時差を考慮して，デイリー・スクラムを開催する．図２に米国がサマータイム期間中におけるコンポーネントB開発チームのデイリー・スクラム実施時間と時差を視覚的に示す（図２中 U.S. は米国に相当する）．他のコンポーネントの開発の場合でも，チーム・メンバーが複数の地域に分散していることが多いが，米国とアジアの国が 1 つのチームで開発を行う場合，チーム・メンバーが相対的に大きな時差となるタイムゾーンに属することになり，それがアジャイル・プロセスの運用を困難にする．

4.分散アジャイル開発における課題

前述の通り，筆者らの開発体制は，1つのチームでありながら，開発拠点が，米国，中国そして日本とグローバルに点在しており,チーム･メンバーはそれぞれ,地理的,時間的に隔てられつつ開発作業を行っている．このような場合，タイムゾーンの違いによってワーキング・タイムが交わらないため，通常のアジャイル開発に比べると，コミュニケーションをとる機会が制限される．

また，デイリー・スクラムは，あるメンバーにとっては早朝であり，またあるメンバーにとっては深夜であるような時間に開催せざるを得ない．デイリー・スクラムは，進捗の評価および問題の確認を通してチーム内のコミュニケーションを活性化させるが，早朝に会議があったメンバーはその日の内に会議の内容に基づき即座に行動しやすい．しかしながら，深夜に会議があったメンバーにとっては，一晩おいた翌朝から行動することになり，デイリー・スクラムで緊急な問題が発覚したような場合は早急に行動をとることが困難となる．

本稿では，問題を 3 つに分類する．

1. 即時コミュニケーションが困難
2. 即時アクションが困難
3. コミュニケーション負担の増大

4.1 即時コミュニケーションが困難

この分類に属する問題の例を以下に列挙する．

- 仕様に変更点がある．
- 仕様に不明点がある．
- ソース・コードに不明点がある．
- 計画項目に漏れがある．

アジャイル・プロセスにおいては開発中に機能の仕様変更が頻繁に発生する．仕様変更が発生すると,チーム･メンバーの作業に影響が出るため，変更の必要性や新たな仕様の妥当性等の確認のために，チーム内でのコミュニケーションが必要となる．チームが 1 つのロケーションに存在し，地理的，時間的な隔たりがなければ，変更発生の都度，関係するチーム・メンバー同士が対面で直接議論を行い，コミュニケーションをとればよい．しかしながら，分散アジャイル開発においては，そのように必要に応じて対面でコミュニケーションを行うことはできないため，異なるコミュニケーションのあり方が必要になる．

仕様やソース・コード等に不明点がある場合，そのような問題点を抱えているメンバーは，疑問を解消し，作業内容を明確化して円滑な開発を続けるために，関係するチーム・メンバーと協議することを望む．しかしながら，上記のような分散アジャイル開発におけるコミュニケーションの難しさがあり，これを妨げている．

また，計画項目の漏れについても同様である．アジャイル開発では，スプリントと呼ばれるサイクルによって作業の計画，実行を繰り返し行う．作業項目はそれぞれのスプリントの最初に行われるスプリント･プランニング･ミーティングですべて計画されていることが望ましい．しかしながら，計画から漏れた作業，急遽必要となった飛び込み的な作業，などが生ずることがある．単一のロケーションのチームでは，開発計画に漏れがあることに気付いたメンバーは，他のチーム・メンバーと即座に連絡を取り合うことで計画の補足を行うことができる．しかしながら，分散アジャイル開発では，コミュニケーションを直ちにとることが難しいため，結果として計画項目の漏れを直ちに修正することが困難なことがある．

4.2 即時アクションが困難

この分類に属する問題の例を以下に列挙する．

- テストの失敗
- ビルドの失敗
- 他のコンポーネントと関連する問題
- マシン障害

開発作業中に問題が発生した場合に，リリース直前などの理由により，問題を即座に解決しなければならないケースが多々ある．そのような場合には，早急なコミュニケーションが必要であり，かつ，問題の解析，エラーの修正等のアクションを直ちにとらなければならない．テストの失敗を発見したメンバーがこれらのアクションを単独で行えればよいが，そのようなケースはまれで，解決には他のメンバーの助力が必要な場合があり，また，テストの実行者とエラーを修正する適任者は異なる場合が多い.4.1 で述べたようなコミュニケーションの困難さによって，他のメンバーの協力を迅速に得ることが妨げられて

いる．例えば，都合よくデイリー・スクラムでテスト失敗の事実と内容をチーム内で確認できた場合でも，エラー修正の適任者がデイリー・スクラムを夜に開催するタイムゾーンに属していると，作業開始が半日遅れ，翌日からあらためてテスト結果の解析を始めることとなる．このような作業の遅延は，開発のフェーズによっては重大な問題となる．ビルドに失敗している場合なども同様の問題がある．

他のコンポーネントと接続，統合する部分に関わる問題の場合は，チーム内での議論だけではなく，他のチームとの議論が必要になる場合がある．普段やりとりのないチーム，メンバーとの議論においては，4.1 で述べたコミュニケーションの難しさに加えて，問題の緊急度の認識がずれていることがあり，議論の進行の遅れは特に顕著である．

また，マシン障害やネットワーク障害等の，開発環境の問題が発生することがある．このような場合には，担当部署に速やかに通知し，事態の改善を要請することになるが，上記のようなコミュニケーションの難しさに起因する対策の遅れにより，開発作業を数日間止めてしまうようなケースも少なくない．

4.3 コミュニケーション負担の増大

通常のアジャイル開発と比べて，分散アジャイル開発の場合，コミュニケーションは，労力を伴う作業となる．4.1，4.2 で述べた各問題は，コミュニケーション当事者間で行われるが，将来的に同様の問題が発生し，他のメンバーがその問題を担当する，または，遭遇する場合，当該メンバーに対して再度，同一のコミュニケーションを行う必要が出てくる．このようなコミュニケーションの重複は，分散アジャイル開発においては大きな負担となる．

5. 分散アジャイル開発における課題に対する解決策

4 章で述べた各問題に対して，コンポーネントB開発チームでは，Rational Team Concert（以下，RTC）［7］をコラボレーションのためのツールとして活用している．RTC は、プランニングやレポートといった開発作業を支援するための充実した機能を持ち，情報をレポジトリーで一元管理することができる．メールなどの非同期コミュニケーションの手段と比べて，コミュニケーションが単一のリポジトリへ集約されるため，過去の議論を確認するために行う情報検索の時間が短縮され，また，公開されたレポジトリーによって，当事者以外のチーム内外のメンバーがコミュニケーションに参加しやすいという特徴を持つ．図 3，4 にコラボレーション・ツール導入前・後のシステム構成を示す（図 3,4 中 U.S. は米国に相当する）．

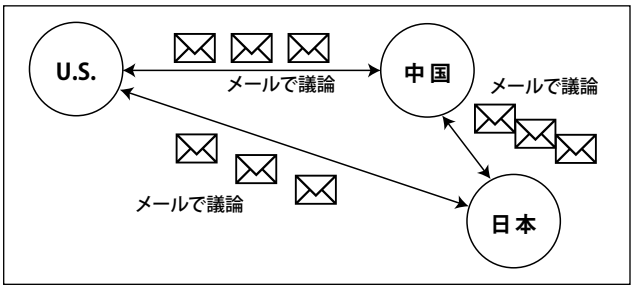

図3. メールによりコミュニケーション履歴が埋もれ,断片化

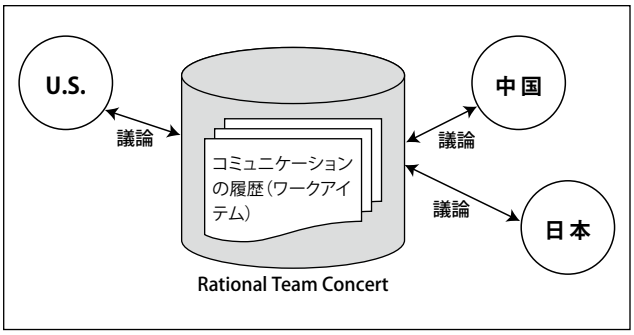

図4. コラボレーション・ツールによりコミュニケーション履歴が集約・公開される

4 章で述べた各課題と，筆者ら日本のチームの取り組み，およびその効果を表 1 にまとめる．以降，それぞれの解決策について順に説明する．

5.1 コラボレーション・ツールを介した非同期，および自発的なコミュニケーションの促進

4.1 で述べた即時コミュニケーションが困難という課題に対しては下記の取り組みが有効である．

- コラボレーション・ツールを介した非同期コミュニケーションの促進
- チャットや電話会議などの同期コミュニケーションの対象を重要または緊急の話題に集約
- コラボレーション・ツール上で行う自発的なコミュニケーションの促進

基本的なコミュニケーションとして，通常のアジャイル開発で行うデイリー・スクラムを電話会議形式で開催する．コミュニケーションを行う必要が生じたメンバーは，その時点で RTC などのコラボレーション・ツールを使用して非同期なコミュニケーションを開始する．コラボレーション・ツールを使用する際は，必要な関係者を通知リストに入れる．関係者として通知を受けたメンバーは，自分の担当ではない場合でも，意見やアドバイスを積極的に

表1. 分散アジャイルにおける課題,および解決策と効果

課題	解決策	効果（メールを用いた場合との比較）
即時コミュニケーションが困難	●コラボレーション・ツールを介した非同期コミュニケーションの促進 ●コラボレーション・ツールを併用した同期コミュニケーションの促進 ●自発的コミュニケーションの促進	●非同期にコミュニケーションする際や，デイリー・スクラムの準備の際，過去の議論を検索する時間が短縮される ●同期コミュニケーションでは適宜，過去の議論を参照するため，間接的に全体の時間が短縮される ●潜在的にメンバーが遭遇する問題についてのヒントが提供されることで，コミュニケーションのための待ち時間半日〜１日が実質０分へ短縮される ●メンバー全員に開発項目を閲覧可能にすることで開発作業漏れの防止
即時アクションが困難	コミュニケーション履歴の共有	●過去のコミュニケーション履歴を検索する時間（数分〜数十分）が短縮される ●断片化された議論を１つに再編集する時間がかかるが，これが短縮される
コミュニケーション負担が増大	ドキュメンテーションの促進	コミュニケーションのための待ち時間半日〜１日が数分（ドキュメントを探す時間のみ）へ短縮される

答えるようにする．異なるタイムゾーンに属するメンバーが遭遇する可能性のある問題に対するアドバイスが前もって自発的に与えられていれば，当該メンバーは，翌日のデイリー・スクラムを待つことなく作業を進めることができる．また，あらかじめ議論を始めている場合，デイリー・スクラムでは，そのような議論が進行中であること，および議論の概要を確認する．議論を効果的に進めるために，各自過去に行った議論を確認して準備する必要があるが，コラボレーション・ツールにより，コミュニケーション履歴が集約されているので，準備の際，必要な議論情報を検索する時間が短縮される．

緊急の仕様変更に対する議論などについては，チャット等の同期的なコミュニケーションが必要になる．デイリー・スクラム時にあらかじめコラボレーション・ツールで進行中の議論の概要と，デイリー・スクラム後に議論が必要であること，を確認する．このときも，上で述べた通り，事前に議論の内容を共有するために情報検索に費やす時間が短縮される．なお，同期コミュニケーションの場合は，議論中に，事前の議論結果を随時参照することがあるが，この際も，適宜，コラボレーション・ツール上で議論済みのアドバイスを参照することで重複した議論を省くことができ，結果として同期コミュニケーション全体にかかる時間を短縮することができる．

作業項目の管理は，コラボレーション・ツールによって一元的に行う．スプリント・プランニング・ミーティングにおいて，スクラム・マスターのリードにより，スプリントで完了すべき作業が洗い出されている状態が理想だが，実際はスプリントの途中で飛び込み的な作業がしばしば発生する．例えば，障害対応やコードレビューなどが挙げられる.各開発者は,上記のコミュニケーション開始のケースと同様に,そのような作業の存在に気が付いた時点で,自発的にコラボレーション・ツール上にタスクを作成する．このように作成されたタスクの計画（どのスプリントで行うか，プライオリティーの設定等）は，必要に応じてチームで行う．これにより，次のスプリントもしくは即時に実行できる．作業の変更が発生した場合はその時点で，情報をコラボレーション・ツールに書き込み共有する．このように，コラボレーション・ツールに作業項目を集中管理し，メンバー全員がコラボレーション・ツールを閲覧することで，開発作業の漏れを防ぐことができる．

5.2 コミュニケーション履歴の共有による円滑な作業の委譲と緊急度の共有

4.2 で述べた即時アクションが困難という課題に対しては，下記の対策が有効である．

- コミュニケーション履歴の共有による問題背景の共有のための時間の短縮
- コミュニケーション履歴の集約による履歴の再編集時間の短縮

コミュニケーションに伴いアクションが必要になるケースでは,作業を他のチーム･メンバーに委譲したり,あるいは,他のスクラム・チームのメンバーの協力が必要になる場合がある．この場合もコラボレーション・ツールの活用が有効である．

コラボレーション・ツールを利用していると，担当者を新たに追加した場合でも，それまでのコミュニケーションの経過が履歴として残っているので，途切れることなく作業を移行することが可能になる．例えば，緊急の問題をデイリー・スクラムで確認した際に，デイリー・スクラムに夜に参加するメンバーが担当している場合，朝に参

加するメンバーに作業を円滑に委譲する必要がある．メールベースのコミュニケーションの場合，過去の議論を検索するための情報検索に時間がかかり，かつ，断片化された議論を新たなメンバーにもわかるように再編集するための時間が必要になるが，それまでのコミュニケーション履歴がコラボレーション・ツールで管理されていると，委譲される側のメンバーはコラボレーション・ツール内のコミュニケーションの履歴を見ることで問題の把握を迅速に行うことができ，背景共有のための時間を短縮することができる．

ビルド・エラーなど，他のチームとの連携作業が発生したような場合でも，そうしたメンバーにコラボレーション・ツール内の履歴を提示することで，緊急度の共有を行うことができる．また，RTC ではコミュニケーションの履歴から，さまざまな情報へとリンクを張ることができるようになっており，ビルド・エラー結果など，適宜必要な情報へリンクをはることで，コミュニケーションが断片化されず，まとまった形で蓄積される．しかし，他のチームとの協業の場合には，普段頻繁にコミュニケーションをとるわけではないため，緊急度の共有に失敗し，コミュニケーションの回答が得られないこともある．その場合，問題の緊急度等を適切に伝えるために，チームリード，スクラム・マスター，マネージャーなどを経由して二重にチェックすべきである．他のチームが異なるタイムゾーンに属している場合，同じタイムゾーンのチーム・メンバーを関係者としてコラボレーション･ツールの通知リストに登録してフォローを依頼する．

5.3 自発的なドキュメンテーションによる知識の共有

4.3 で述べたコミュニケーション負担の増大という課題に対しては下記の対策が有効である．

- 潜在的に作業中断を引き起こす問題についてドキュメンテーションによる知識の共有．

分散アジャイル開発においては，必要に応じて積極的にドキュメント作成を行う必要がある．コミュニケーションは重要な問題のために行い，それ以外は，ドキュメンテーション，および，その参照先を閲覧することなどで解決する．これにより，コミュニケーションの重複を減らすことができる．具体的には下記の項目をドキュメント化することを推奨する．

- 開発における作業履歴を残す．他人の作業結果をいつでも参照できる環境になっていることは，自発的なコミュニケーションを促すためには重要である．これにより，チーム・メンバーの出入りが頻繁に行われるような場合でも，作業が途切れることなく引き継ぐことが可能になる．
- 開発におけるヒント，いわゆるチップス的なものをドキュメント化することも重要である.自分にとって障害であった点は他の開発者にとっても障害となる．筆者らの場合は，RTC 内にバンドルされている Wiki にテスト・マシンのセットアップ方法の情報を共有している．
- 外部の技術コミュニティーへの情報発信も重要である．IBM の場合，IBM developerWorks［8］などのサイトを介して，ユーザと製品開発者がコミュニケーションを行う．このような場合，コンポーネントの担当者があらかじめ，developerWorks に記事として情報を提供していると，ユーザにとっては質問する手間が省略されるし，製品開発者はその都度，対応する手間が省略される．

6. まとめ

本稿ではグローバルに分散したチームでアジャイル開発を行う際の課題とその解決策について述べた．課題としては，地理的・時間的に分散した地点にいる開発メンバー間でのコミュニケーションを取り上げた．その際，課題の本質は，タイムゾーンが異なるために，ワーキング・タイムが交わらず，コミュニケーション機会が制限されること，タイムゾーンが夜のメンバーが即時アクションを取ることが困難であること，という二点にあることを確認した．次に，コラボレーション・ツールを導入することで，非同期・自発的なコミュニケーションを促進するアプローチを提案し，その効果について述べた．非同期コミュニケーションについては，メールを用いたときと比べて，過去のコミュニケーション履歴を検索する時間，それらを他のメンバーへ伝えるためのつなぎ合わせる等の再編集のための時間，自発的なコミュニケーションの促進による潜在的な問題への回答の提供など3つの効果があった.しかしながら,デイリー・スクラムで日々の進捗を確認する場合や，緊急度の高い話題を議論する場合などは，電話会議やチャットを用いている．こうした同期コミュニケーションのための手段は依然として重要であり，RTC などのコラボレーション･ツールによって置き換えるというよりは，同期コミュニケーション中に過去の議論を適宜参照するという形で併用する形が望ましい．本稿が提案する解決策により，プロジェクトAを通じて，グローバルに分散した数十人規模のチームでもアジャイル開発が機能することを確認した．

本稿は，単一のプロジェクトに関する知見を元にア

プローチを提案しているため，今後は，他のプロジェクトにおいて，4 章で扱った課題に漏れがないか，5 章で提案したアプローチが筆者らのプロジェクトと同様に有効に働くことを検証する必要がある．

本稿では言及しなかったが，アジャイル開発ではウォーターフォール開発に比べて，継続的なビルドなど，コミュニケーション以外にも大きく変化している取り組みがある．今後の検討課題としては，こうした取り組みが，分散アジャイル開発にどのように適用可能かを分析し，適切なアプローチを検討する必要がある．

謝辞

論文執筆に際して，IBM 東京ソフトウェア開発研究所の若尾正樹氏にアドバイスをいただきました．あらためて深謝いたします．

参考文献

［1］ IPA: 非ウォーターフォール型開発の普及要因と適用領域の拡大に関する調査 , http://sec.ipa.go.jp/reports/20120611.html
［2］ Wikipedia: ウォーターフォール・モデル , http://ja.wikipedia.org/wiki/ ウォーターフォール・モデル
［3］ Wikipedia: アジャイルソフトウェア開発：http://ja.wikipedia.org/wiki/ アジャイルソフトウェア開発
［4］ アジャイルソフトウェア開発宣言 , http://agilemanifesto.org/iso/ja/
［5］ アジャイル宣言の背後にある原則 , http://agilemanifesto.org/iso/ja/principles.html
［6］ Dean Leffingwell（著），玉川憲ほか（訳）：アジャイル開発の本質とスケールアップ , 翔泳社（2010）
［7］ IBM：Rational Team Concert, http://www.ibm.com/software/jp/rational/products/scm/rtc/
［8］ IBM: IBM developerWorks, http://www.ibm.com/developerworks/jp/

日本アイ・ビー・エム株式会社
東京ソフトウェア開発研究所　ラショナル開発
スタッフ・ソフトウェア・エンジニア

熊谷 賢 Ken Kumagai

［プロフィール］
2006 年日本 IBM 入社．2008 年半ばまで，主に JavaScript を用いたアセット開発を担当．2008 年半ばより，ラショナル製品の開発を担当．

日本アイ・ビー・エム株式会社
東京ソフトウェア開発研究所　ラショナル開発
アドバイザリー・ソフトウェア・エンジニア

堀内 芳雄 Yoshio Horiuchi

［プロフィール］
1990 年日本 IBM 入社．ソフトウェア製品開発，システムの国際化などに従事．2003 年よりラショナル製品の開発を担当．電子情報通信学会会員．