Technology Reports

アジャイル　リモートワーク　アプリ開発

アジャイルアプリ開発体制の効率的なリモートワーク環境への移行事例

移動機開発部　福囿 貴嗣（ふくぞの たかし）　大貫 詠子（おおぬき えいこ）
若山 希樹（わかやま きき）

アジャイル開発においては，メンバが同一拠点に集合し作業を行うことが一般的であるが，新型コロナウイルス感染拡大を回避するため，リモートワークを取り入れた開発体制が必要となる．そこで効率を落とさず開発を行うために，ドコモでは，Webタスク管理・自動化ツールやWeb会議など関連ツールを組み合わせてリモートワークを行うようにした．これにより，品質を保ちかつコロナ禍以前と変わらない短周期でのリリースを可能とした．

1. まえがき

ドコモテレビターミナルは，ドコモが販売しているテレビ向けデバイスである．TVに接続することでひかりTV for docomoをはじめとしたさまざまな映像サービスを視聴可能である．

ドコモテレビターミナルアプリは，ドコモテレビターミナルを操作するスマートフォン用のアプリであり，ドコモテレビターミナルのリモコン操作や録画コンテンツの持出しなど数多くの機能がある．また，ドコモテレビターミナルアプリは，市場環境の変化に合わせて次々とバージョンアップする動画サービスに対応するため，優先度をつけ必要な機能から短周期に開発を行うアジャイル方式での開発が有効である．

アジャイル開発では，短周期な開発を行うため，コミュニケーションを密にとれる環境が必要であり，同一拠点で開発を行うことが一般的である．ドコモテレビターミナルアプリにおいても開発チームを1拠点に集め，コミュニケーションが密にとれる環境で開発を行ってきた．

ところが，首都圏において新型コロナウイルス感染者が増加したため，アジャイル開発チーム内における新型コロナウイルス感染のリスクを踏まえ，2020年2月から開発ツールやコミュニケーション手段などの整備を実施した上で，原則開発メンバはリモートワークにて開発を行うこととした．

本稿では，ドコモテレビターミナルアプリのア

ジャイル開発の実践事例，および開発のリモート化に伴う工夫点について解説する．

2. アジャイル開発概要

2.1 アジャイル開発のフレームワーク

アジャイル開発とは，短期間で開発を繰り返すソフトウェア開発手法の総体であり，アジャイル開発で名声のある人たちが「アジャイルソフトウェア開発宣言」という形で，アジャイルソフトウェア開発に共通する価値観と行動原則を宣言したことが始まりである［1］．

アジャイル開発は，短期間での開発を繰り返して進めていくため，ウォーターフォール開発*1に比べ変化に強いという特長があり，市場のユーザニーズの変化にいち早く柔軟に対応できる．

アジャイル開発には，スクラムやエクストリーム・プログラミング*2，カンバン*3などのさまざまなフレームワークがあり，我々のプロジェクトではその中でも軽量で理解しやすい（が，習得はなかなか難しい）スクラムというフレームワークを採用し開発を進めてきた．

スクラムは，「スクラムガイド」に理論や定義が記載されており［2］，適宜内容の改定が行われている．スクラムは，チームが正しく進捗しているか確認する「検査」，検査の結果プロセスの改善を行う「適応」，現在の状況や問題点などがすべて見える化されている「透明性」の3つを原則とし，3つのロール，5つのイベント，3つの成果物が定義されている．

2.2 スクラムのプロセス概要

(1)スクラムにおける3つのロール（**図1**）

①プロダクトオーナー（PO：Product Owner）：スクラムチームから生み出されるプロダクトの

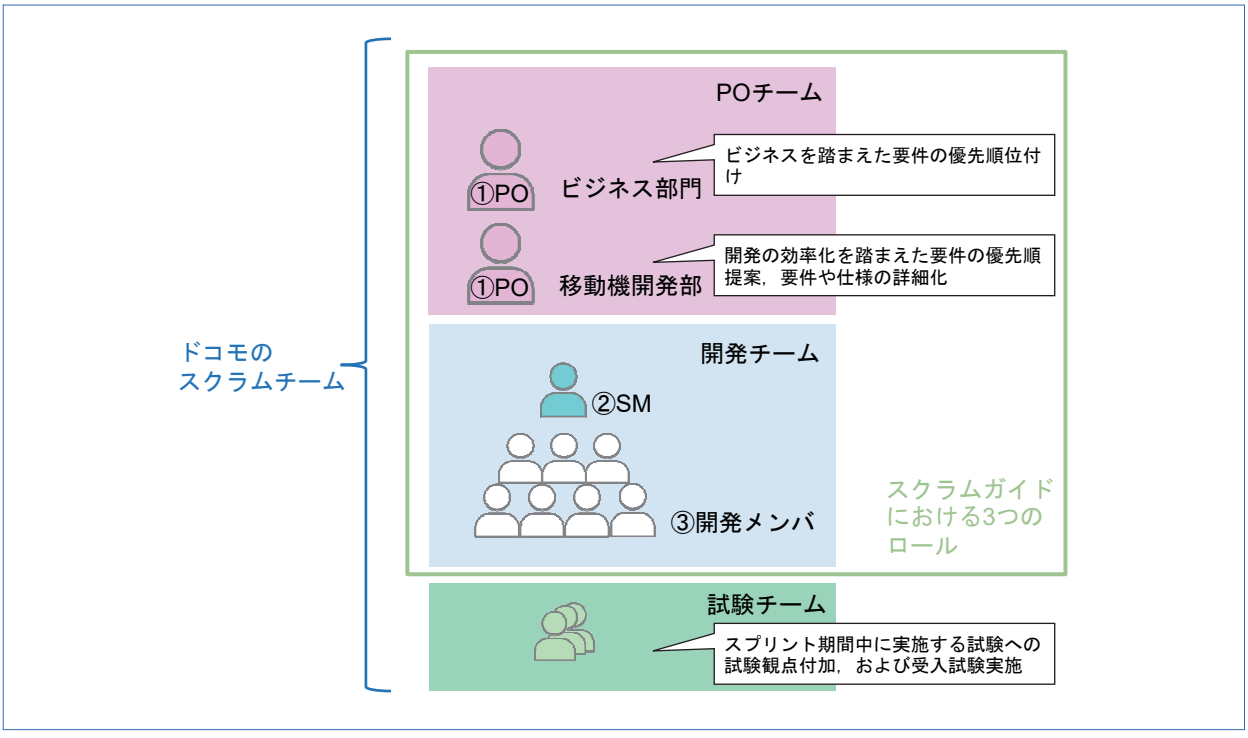

図1　体制図

*1　ウォーターフォール開発：要件定義，設計，実装，評価を工程ごとに順序立てて実施していく開発手法．

*2　エクストリーム・プログラミング：アジャイルソフトウェア開発手法の一種で，効率的に開発を行うための経験則を極端（エクストリーム）なまでに実践する手法．

*3　カンバン：アジャイルソフトウェア開発手法の一種で，作業項目を「かんばん」というリストで可視化することで，作業を継続的に実施・改善する手法．

価値を最大化することに責任をもつ．プロダクトの責任者であり，プロダクト全体の機能ごとの要件を記載したプロダクトバックログを作成し，優先順位付けを行う．

②スクラムマスター（SM：Scrum Master）：スクラムの理解と実践を推進し，プロセスがうまく回るようにすることに責任をもつ．チームの進捗を妨げるものを排除したり，チームが改善できるよう手伝ったりすることで自己管理型および機能横断型のチームに導き，スクラムチームが有効に機能するように取り計らう．

③開発メンバ：開発メンバはプロダクトの実現者であり，上下関係はない．全員が揃うことでプロダクト作成能力が満たされる．POと合意したプロダクトバックログ項目を完成させるよう最大限努力し，常に改善をする責任をもつ．

⑵スクラムにおける5つのイベント

①デイリースクラム：開発チームの状況を3つの質問（昨日やったこと，今日やること，困っていること）で毎日検査する場．最長15分で延長無し．

②スプリント[*4]レビュー：開発チームのスプリント成果物（アプリ）をPOが確認する場．関連のあるステークホルダにも披露する．その後，フィードバックをもらい，見直すこともある．

③スプリントプランニング：計画会議とも呼ばれ，そのスプリントでの開発計画を立てる．POがプロダクトバックログのWhat，Whyを話し，開発チームがHowを話す．また，開発チームはこのスプリントで，プロダクトバックログの要件をどの程度達成できそうかを話す．

④スプリントレトロスペクティブ：振返りとも呼ばれ，さらにうまく仕事を進められるように改善を繰り返すためのイベントである．そのスプリントでうまくいったこと，今後の改善点を整理する．バグを直すのではなく，バグが生まれるプロセスを直す．我々のプロジェクトでは，KPT法[*5]を利用し，メンバへこの後のスプリントで継続すること，改善することの共有・合意を行っている．

⑤バックログリファインメント[*6]：次回以降のスプリントに向け，プロダクトバックログの手入れをする．

⑶スクラムにおける3つの成果物

①プロダクトバックログ：プロダクト全体の機能ごとの要件を記載したもの，いわゆる欲しいものリストのことである．価値の高い機能から優先順位をつけて並べ，リストの上位から順に開発する．優先順位は常に更新し，最新のものにする必要がある．

②スプリントバックログ：プロダクトバックログを具体的なタスクに分割したものである．

③スプリント成果物：スプリント単位で開発チームが作成したものであり，リリース判断が可能な成果物である．アプリ開発では，動くアプリとドキュメントにあたる．

2.3 本プロジェクトの体制

我々のプロジェクトでは，ビジネス部門および移動機開発部がPOチームとしてスクラム開発を進めている．開発の効率化を踏まえた優先順位の提案を移動機開発部が行い，最終的にビジネスを踏まえた優先順位付けをビジネス部門が実施するという役割分担となっている．また，開発委託先のメンバでSMおよび開発チームを，試験委託先のメンバで試験チームを構成した（図1）．

3. 短周期リリースに向けた工夫

3.1 要件定義フェーズ

⑴開発優先順位付け

ビジネス部門は開発要望に対してビジネス優先度

＊4 スプリント：短く区切られた開発期間のことで，スクラムガイドでは1カ月以内に制限されている．

＊5 KPT法：プロジェクトの進め方に対して振返りを行うフレームワーク．スプリント期間中の取組みや進め方に対して「Keep（良かったこと）」「Problem（課題）」「Try（Problemに対する改善策）」を挙げて振返りを行う．

＊6 バックログリファインメント：スクラムで定義されているイベントの1つ．次回以降のスプリントに向け，プロダクトバックログの手入れをする．

を付与し，優先度の高い開発要望から順に移動機開発部と開発チームで要件の詳細分割を行う．詳細分割した要件をビジネス部門にフィードバックし，開発優先順位の決定を行う．

①ビジネス優先度付与

開発要望の各項目について，ビジネス部門は優先度，およびビジネスにどのような効果をもたらす開発要望なのかを，移動機開発部と開発チームに対して説明する．そうすることで，チーム全体で目的の共通認識をもつことができ，この後の工程で移動機開発部と開発チームが，適切に要件の詳細分割や開発優先度付けを行うことが可能となる．

②要件の詳細分割

ビジネス部門が付与した優先度の高い開発要望から順に移動機開発部と開発チームで要件の詳細化を行い，機能をさらに分割していく．移動機開発部はビジネス効果と開発規模，開発効率から判断して，分割した詳細機能に開発優先順位を付与する．分割した詳細機能と開発優先順位をビジネス部門にフィードバックし，最終的な開発優先順位の決定をビジネス部門が行う．

⑵一気通貫でのチケット*7管理

POチームから開発チームまですべてのメンバに対して，効率的にチケットを更新・共有する取組みを行った（**図2**）．チケット管理ツールに要件（What，Why）を親課題としてPOチームが記載した後，開発チームがどのように実現するかという作業内容（How）を詳細化して子課題として記載する．さらにソースコードレビューツール上で，どのように実装するかの議論を実施し，チケット管理ツールと連携することで，ソースコードの変更箇所がどの要件のどの作業によるものかを一気通貫で管理・把握できるようになった．

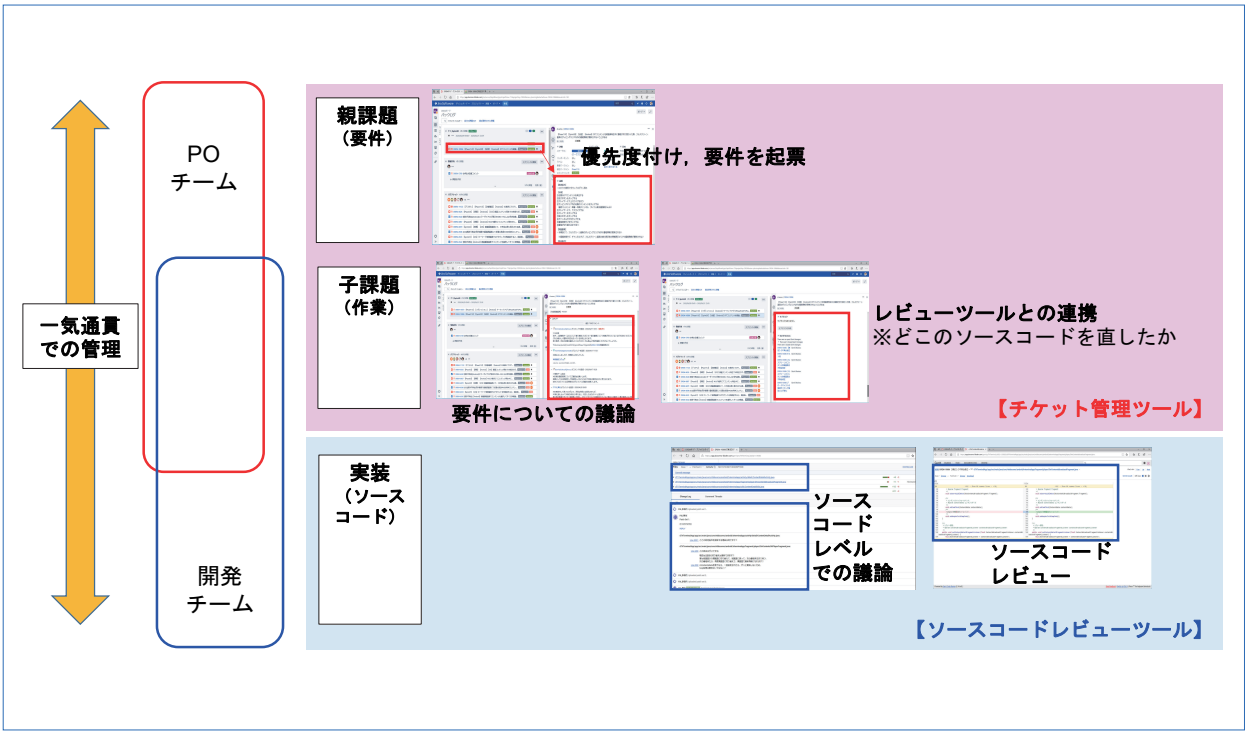

図2　一気通貫でのチケット管理

＊7　チケット：開発プロジェクトで発生した作業や課題などのタスク．

結果として，要件・作業・ソースコードの修正箇所における認識違いや漏れが起こりづらくなり．手戻りが減少した．

①要　件

要件の段階では，チケット管理ツールになるべく詳細に内容を記載するだけではなく，なぜその開発要望が出されているのか，という背景や目的についても注力してPOチームが記載を行うこととした．結果として具体的な作業に開発チームが落とし込む際に，各自がビジネス的な目線も踏まえた設計の検討が能動的に実施できるような環境が整えられた．

②作業内容

親課題を要件とし，開発チームが要件を踏まえた上で，ソフトウェアの実装に必要な作業内容を子課題として記載する．要件と作業内容とをリンクさせることで実装内容に漏れがないことを担保した．

さらに，作業内容とソースコードレビューツールとを連携させることでソースコードの修正箇所とマッピング，あとから振り返った際に経緯を把握することを可能とした．

③テキストチャットツールでの補完

軽微なQ＆Aなどについてはチケット化を行わず，テキストチャットツールでQ＆Aを実施する．テキストチャットツールは，メールやチケットツールと比較してあいさつ文などは不要であり，内容の分類などを厳密に行わずとも質問可能という特長がある．そのため，記載時間などのコミュニケーションコストが削減され，また詳細について決まっていない段階でもチャットを続けるうちに内容が固まってくるという効果もあり，チケット化する前の意識合せやチケット内容の詳細化などで利用している．

3.2　開発フェーズ

開発フェーズにおいては，個々の作業用端末に開発に必要なドキュメントやソースコードなどの情報を分散させず，クラウド上に情報を集中させ，メンバに共有できる仕組みを構築した．また，ビルド後の署名付与など連続して実行が必要なツールと連携して，ビルドを定期的に自動実行できるようにした．

⑴開発ツールのクラウド化

①ソースコードのクラウド管理

ソースコードについてはクラウド上に保管を行い，各自の作業用端末からロケーションを問わず容易にアクセスできるようにすることで，同じソースコードを開発チーム全員で共有できるようにした．また，商用ブランチ*8や開発ブランチを明示的に表示することで，リグレッション*9などを防ぐ工夫を行った．

②レビュー回覧ツール

ソースコードについて，開発チーム有識者でレビューを行うが，広くチーム内の意見が確認できるように誰が回覧し，どこを指摘し修正したかのログが追えるようなツールを導入した．これにより修正箇所の漏れをなくし，またレビューの進捗をチーム内で十分に管理できるようになった．

③ソースコード解析や難読化*10

ビルド*11前にソースコードの静的な構文解析*12を実施しているが，各自の作業用端末で実施すると結果が共有されない．そこで，解析結果をクラウド上に表示し各人が確認するフローとすることで結果の横展開がされるようになった．

また，リバースエンジニアリング*13による解析を避けるために商用提供前にアプリを難読化しているが，難読化漏れが発生しないよう，クラウド上のツールで難読化を施し，結果をログとしてサーバに残すことで，難読化実行状況

---

*8　ブランチ：バージョン管理を行う際に，マスターとする履歴から分岐して記録すること．例えば，商用提供されているバージョンを商用ブランチ，まだ商用提供されておらず開発中のバージョンを開発ブランチなどと称する．

*9　リグレッション：ソフトウェアをバージョンアップした際に，性能が低下したり，機能などが改悪されたりすること．主に過去の不具合が復活することで生じる．

*10　難読化：悪意のあるユーザによるアプリの解読や改ざんが困難となるよう，ソースコードに対しプログラムの動作を変えずに人間に理解しづらい加工を行うこと．

*11　ビルド：ソースコードを基に，端末上で実行可能なファイルや配布パッケージを作成する処理や操作．

が開発チームメンバで共有できるようになった．

⑵ツールの自動化

①ビルドの自動実行

一般的には開発チームメンバが，必要に応じて個々にアプリをビルドすることが多いが，その場合，メンバによって検証を行うバージョンが異なってしまうなどの課題がある．そのため，システム上で定期的（具体的には1日1回夜間）に自動でビルドを実行し，そのアプリを最新版として翌日作業を行うというルールを導入した．チーム全体が最新版のアプリを共有することができ，古いアプリで検証作業を進めることによるリグレッションの発生も無くなった．また，変更箇所の履歴をサーバ上のログとして残し，共有することで，ソースコード変更の気付きを開発チーム全体に与えるとともに，変更箇所の見逃しが減少した．

②署名自動化

ドコモがサービスを提供しているアプリであることをユーザに明示するために，アプリに対して署名を付与している．今までは開発チームがビルドしたアプリを，移動機開発部が署名付与サーバにアップロードし，署名付与を実施していた．しかし，開発チームとのファイルの受渡しなどの手間をなくし迅速に開発を行うために，クラウド上に商用ブランチとして保管されているソースコードのビルドを移動機開発部が実施するようにした．さらに，ビルドを実施した後，自動で署名付与サーバにアプリを送信して署名をアプリに付与するスクリプト＊14を作成することで，手動での署名付与作業を削減した．

3.3 試験フェーズ

従来は，すべての要件の開発が完了した最終スプリント後に，受入試験としてすべての要件の動作確認を行い，マーケットリリース判断を行っていた．そのため，受入試験にかかる時間が長く，開発終盤で検出する不具合が多く存在した．このやり方では手戻りが発生し，マーケットリリースまでに不具合修正の時間がかかってしまい，短周期リリースは実現できない．

そこで，不具合をなるべく早い開発段階で検出できるように，受入試験でまとめて実施していた一部の試験を，スプリント期間中の前工程で実施することとした．

⑴ユーザシナリオ試験

図3(a)に示すとおり，試験フェーズ改善前は全機能実装が完了した最終スプリント後にまとめてユーザシナリオ試験を実施していたため，試験にかかる時間が長く，開発終盤に不具合が多く検出されていた．そこで，スプリントごとに，開発した要件のユーザシナリオ試験を実施する方針に変更した（図3(b)）．その結果，開発の早い段階で不具合を検出し，修正することが可能となった．また，全機能実装後の要件確認を探索的試験＊15に変更することで試験にかかる時間の短縮が可能となった．

⑵リグレッション試験

リグレッション試験もユーザシナリオ試験同様に，試験フェーズ改善前は全機能実装が完了した最終スプリント後に試験を開始していたことで，開発終盤にて不具合が検出されている状況だった（図3(a)）．

そこで，各スプリント完了時に成果物に対してリグレッション試験を実施する方針に変更した．試験工数を考慮して，リグレッション試験については試験の自動化も併せて行った．その結果，開発の早い段階で不具合を検出し，修正することが可能となり，開発終盤で不具合を検出することはほとんどなくなった（図3(b)）．

⑶効　果

各スプリント完了時に，成果物が市場品質レベルまで上がっていることを確認済みのため，全機能実

＊12 **構文解析**：ソースコードを解析して，ルールや文法に違反している箇所を発見すること．

＊13 リバースエンジニアリング：ソフトウェアの動作を観察したりソースコードを解析したりすることで，当該アプリケーションの技術情報を調査し明らかにすること．

＊14 スクリプト：単純な処理を行うプログラムを記述するための，簡易プログラミング言語をスクリプト言語といい，そのスクリプト言語で記述されたプログラムをスクリプトと呼ぶ．

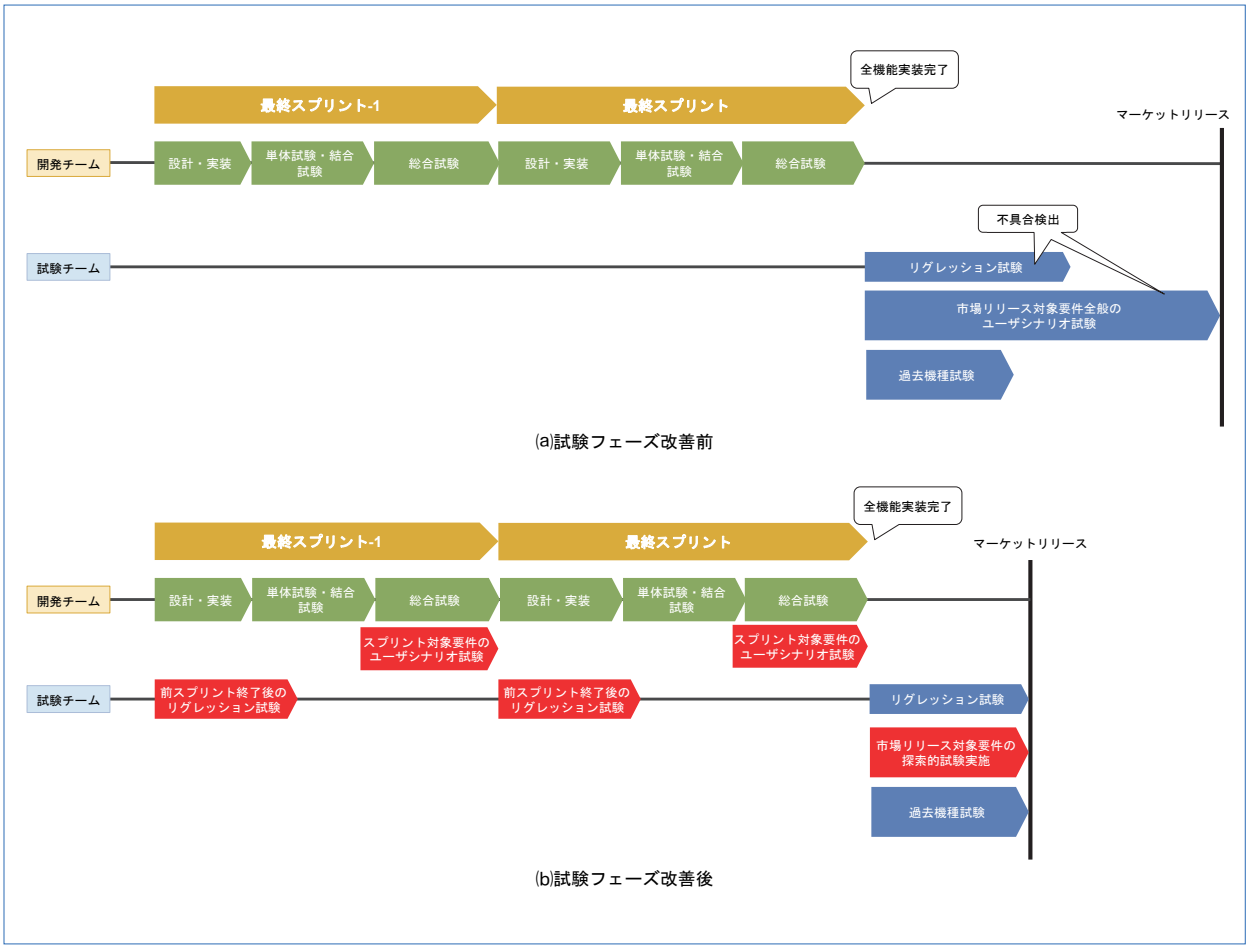

図3　試験フェーズ改善による試験時間短縮

装完了後は，マーケットリリース前の最終品質確認として最低限の動作確認のみ実施した．以前は，全機能実装後の試験に平均1カ月程度かかっていたが，3営業日程度まで短縮できた．

4. 開発のリモート化に伴う変化

4.1　概　要

アジャイル開発チームで新型コロナウイルスの集団感染が発生するリスクを考慮し，リモートワークでの開発に移行した．その際，各種スクラムイベントも対面前提からリモート開発に適した手法に変更した．コミュニケーションやセキュリティなどリモートワーク特有の課題もあったが，生産性を落とさない工夫を行うことで円滑な移行を可能とした．

4.2　リモートワーク移行事例

(1)各種スクラムイベント

①デイリースクラム

リモートワーク移行前は，全メンバがチケット管理ツールを表示した大画面モニターの前に集合し，各メンバの進捗を確認していた．リモートワーク移行後はWeb会議システムを導入し，チケット管理ツールを画面共有しながら

＊15 探索的試験：事前にテストケースを作成するのではなく，ソフトウェアの振舞いやテスト結果を確認しながら試験を実行する手法．事前の試験設計やパターン網羅的な試験を行わないため，従来の記述式試験と比べてスピーディに実施でき，アジャイル開発との相性がよい．

各メンバの進捗を確認することで，リモートワーク移行前と同様にデイリースクラムが運用できるようにした．

②スプリントプランニング，スプリントレトロスペクティブ

この2つのイベントについては，メンバ同士の議論が多いスクラムイベントのため，顔を見せた状態でWeb会議を実施している．リモートワーク移行初期の頃は顔を見せない状態でWeb会議を実施していたが，お互いの反応や考えが音声だけでは読み取れないことから，不安感やモチベーションへの影響もあり，段々と活発な議論ができなくなってきていた．

そこで，原則顔を見せた状態でWeb会議に参加することをルール化した．お互いに顔を見せ合うことで，音声以外の情報である表情，ジェスチャー，視線などから視覚的な付加情報を得ることができる．その結果，リモートワーク移行前と同程度まで活発な議論ができるようになった．

スプリントレトロスペクティブについては，リモートワーク移行前は付箋と模造紙を使いながらKPT法を実施していた．リモートワーク移行後は物理的なツールが利用できないため，スプレッドシート＊16を代用することで，リモートワーク移行前と同様にスプリントレトロスペクティブが運用できている．また，スプレッドシートを利用することの副次的効果として，電子化されているため過去経緯の確認が容易となった．

③スプリントレビュー

リモートワーク移行前においては，スプリントレビューの場で実機を使った動作確認を行い，その場でPOチームが受入判断を行っていた．一方，リモートワーク移行後ではテスト配信ツールを使ってスプリント成果物であるアプリを配信し，各自が自宅で実機にアプリをダウンロードして動作確認ができるようにしている．また，デモ動画を準備し，スプリントレビュー内で流すことでメンバ間の認識齟齬が起こらないように工夫している．

これらの取組みによって，リモートワーク移行前と同様に，スプリントレビューの場でPOチームが受入判断を行うことが可能となっている．

④バックログリファインメント

リモートワーク移行前は対面で実施していたバックログリファインメントについても，リモートワーク移行後はWeb会議システムを導入することで，移行前と同様に運用ができている．

⑵その他

①スクラムイベント外のコミュニケーション

ボイスチャット機能を有するツールを利用することで，勤務時間中はチーム全員が常時音声接続状態にしている．そうすることで，少しの疑問点が発生した場合にも気軽に声がけができ，リモートワーク移行前の同一拠点のときに近い状態のコミュニケーションが可能となる．

また，音声以外のリアルタイムコミュニケーション方法として，テキストチャットも利用している．操作手順や不具合原因など複雑な説明が必要な際には，音声コミュニケーションの補助ツールとしてテキストチャットを利用するとコミュニケーションが円滑に進む．リモートワーク移行前ではホワイトボードを使いながら意識合わせをしていたが，リモートワーク移行後ではテキストチャットを代用している．

②ワークショップ

これまで，定期的に開発に関する課題を各グループに分かれてディスカッションし，それを開発にフィードバックするというようなワークショップを開催することで，メンバの関係性を

＊16 スプレッドシート：Webアプリケーションの一種．複数人による同時編集ができ，リアルタイムで誰がシートのどの部分を編集しているか分かるため，付箋や模造紙の代用として効率的に作業が可能．

深め，自己組織化[*17]を進めてきた．リモートワーク移行後は対面での開催が難しくなったため，拠点開発時に培った関係性がリモートワーク移行後も活かされて，忌憚なく意見を言い合えるような雰囲気を醸成することにより，リモートワーク下でも生産性を維持することに繋がっている．

今後，メンバの入替えが発生した場合も引き続き関係性を維持できるように，リモートワークでのワークショップ開催も検討している．

③セキュリティ

リモートワークでの開発をする上で，重要となるのが情報セキュリティのリスクである．拠点に集合して開発していた際には，その居室に入るためにIDカードが必要で，プロジェクトに関係のない人は出入りができない環境であったが，在宅の場合，家族などこのプロジェクトに関係しない人も出入りできる可能性のある環境で開発を実施することとなる．物理的な対策も同様だが，一定のルールを設け，開発チームのメンバがそのルールを確実に守っていくことが重要になる．総務省から出ているテレワークセキュリティガイドラインでは，「ルール」「人」「技術」のバランスが取れた対策の実施が推奨されている［3］．我々のプロジェクトでも，早期から開発チームと連携し，以下のとおり情報セキュリティリスクへの対策を検討してきた．

(a)セキュリティ運用ルールの策定

運用ルールの策定にあたっては，本プロジェクトに携わる各社のセキュリティ部門が設定しているガイドラインの内容を参考にまとめた．

具体的には，隔離された個室などセキュリティの保たれた環境で作業を行うことや，必ず貸与されたPCを使用し，個人のPCの使用は禁止とすること，フリーソフトのインストール禁止，宅外への持出し禁止などである．

これらの運用ルールを1項目ごとにチェックリスト化した．最初にチーム全体で読合せを行い，その後は月に1回，各自にチェックしてもらうことで，個人への意識付けを定期的に行うとともに，適切に運用されていることを確認している．

(b)ソースコードとドキュメントの管理

ソースコードはクラウド上に保管している．クラウド上の開発環境へは，個人に割り振られたIDとパスワード，そしてクライアント証明書での認証をした上で，VPN（Virtual Private Network）接続[*18]でのアクセスとすることで，セキュリティを保つことができている．

仕様書や設計書などのドキュメントについては，開発ベンダ拠点のサーバに格納し，その拠点のPCを自宅から遠隔操作するリモートデスクトップ方式を利用してサーバの情報を閲覧，編集するルールとし，自宅への物理的な持出しを禁止した．

(c)デバイスの管理

リモートワークでの開発にあたり，個人で購入したPCの使用を禁じ，会社から貸与されたPCを利用するよう決定した．また，本アプリの開発においては，各種スマートフォン・タブレットなどによる動作確認が必要になる．そのため，スマートフォンなどのアプリの確認用のデバイスも各自持ち帰ることが必要であった．これらのデバイスの管理としては，適切に運用されているかのチェックのため，毎朝ビデオ通話でPCおよびデバイスの所在と電源が入るか，などの目視チェックを実施している．これにより，紛失・故障リスクの低減，また万が一，紛失・故障をしたときに迅速な対応ができる．

＊17 **自己組織化**：チームにマネージャーを置かず，メンバが自律的にプロジェクトを運営すること．メンバが自ら計画を立て，日々の進捗を共有し，課題を解決していく．

＊18 **VPN接続**：インターネットなどの公衆回線網上で，認証技術や暗号化などの技術を利用し，保護された仮想的な専用線環境を利用し接続する方法．

5. アジャイルリモートワーク化における生産性と品質

5.1 生産性

拠点に集合して開発していたときと，リモートワークへ移行してからのチームの生産性について，検証した．

⑴ベロシティ推移

スクラム開発では，スクラムチームがスプリント中に完了する平均作業量をストーリーポイントで相対的に表したベロシティという数値がある．ベロシティは，次のスプリントでどのくらいの作業（チケット）が完了できそうか検討する際の参考値としたり，チームの成長度合いを把握したりするために利用される．

ドコモテレビターミナルアプリの開発チームにおけるベロシティおよび，1人当りの対応ストーリーポイントの推移を**図4**に示す．本プロジェクトではリモートワークへの移行と同時期に開発チームの人数に変更があったため，チーム全体のベロシティではなく，1人当りの対応ストーリーポイントにてリモート化前後の推移を確認した．リモートワークへの切替直後は一時的に落ちこんだがその後は回復し，リモート化前と同水準で推移しており，チームとしての生産性を落とさずに稼働できていたことが分かる．

生産性を落とさずに稼働することができたため，アプリのリリース日程やリリース頻度への影響もなかった．

⑵幸福度推移

幸福度とは，個人がどのくらい幸福であると感じているかを作業内容や役割，働きやすさなどを踏まえて5段階でスコアリングしたもので，組織が活気に満ちて働けているかの指標となる．我々のプロジェクトでは，スプリントごとに開発チームのメンバへ幸福度をヒアリングし，アベレージを算出して傾向を分析している（**図5**）．

リモートワーク移行直後は，環境変化への戸惑いの声が多く，一時的に落ち込んだがその後回復し，在宅での開発に関しては，家族と過ごす時間が増えたなどのポジティブな意見を多く聞くことができた．アジャイル開発では，チームの幸福度が高いほど生

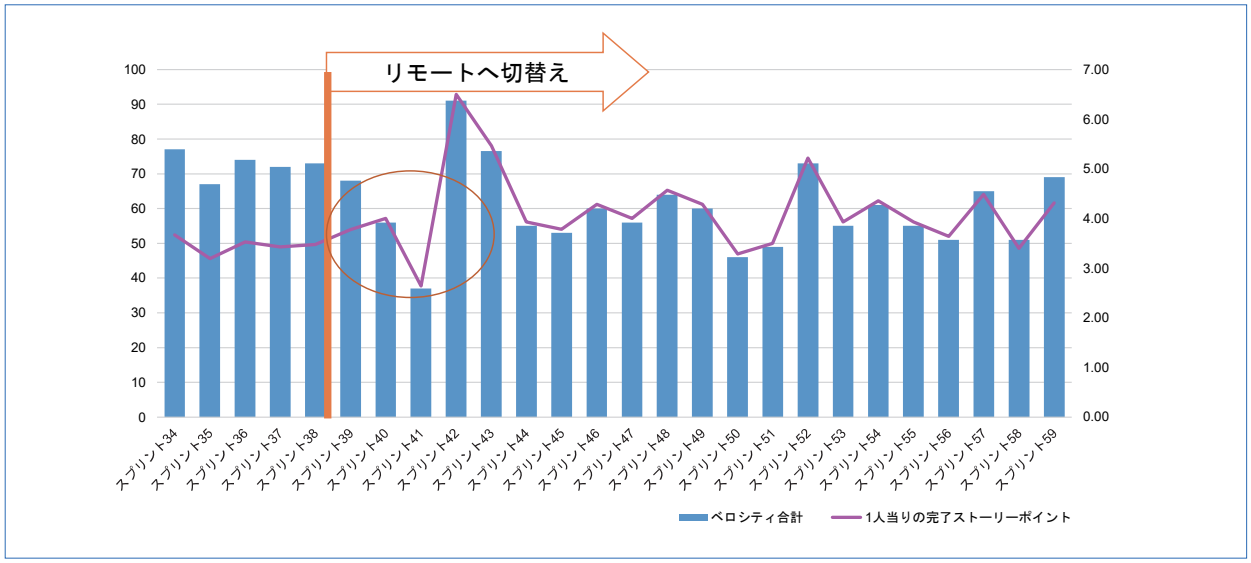

図4　ベロシティ推移

産性が向上するといわれており，幸福度の推移をみると前述で示したベロシティの推移とも類似していることが分かる．

5.2 品　質

リモートワークへの移行前と移行後でアプリの品質に変化があったかどうかについて検証するため，開発フェーズで混入したkStep数当りのバグの摘出率推移を**図6**に示す．本アプリでは1件／kStep以下を品質目標としているが，すべてのスプリントにおいて1件以下を達成しており，リモートワーク切替え前後を比較しても品質の低下はないものと見るこ

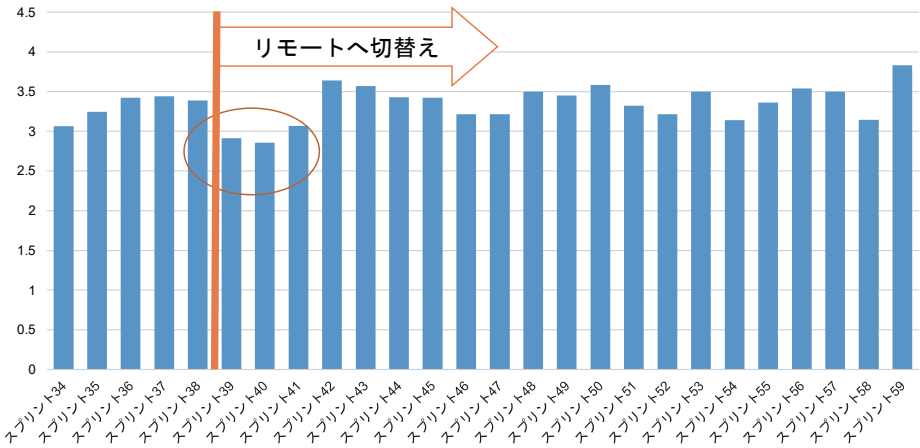

図5　幸福度推移

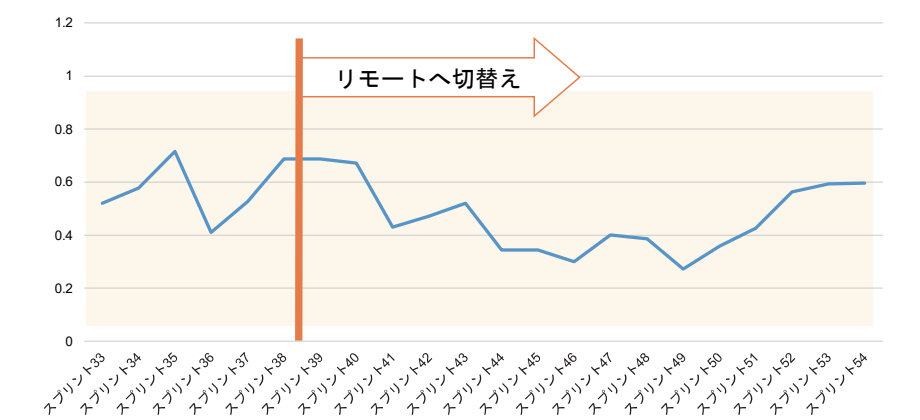

図6　開発フェーズで混入したkStep数当りのバグの摘出率推移

とができる.

6. あとがき

本稿では，アジャイル開発プロジェクトの効率的なリモート開発への移行の事例について解説した．リモート開発においても短周期リリースおよび品質と生産性の維持を実現することができた．アジャイル開発は常に改善を続けていく手法であり，より効率的な開発の実現に向けて今後も引き続き改善の取組みを行っていきたい.

文　献

[1] アジャイルソフトウェア開発宣言.
https://agilemanifesto.org/iso/ja/manifesto.html

[2] Ken Schwaber and Jeff Sutherland：“スクラム公式ガイド：ゲームのルール，”Nov. 2020.
https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Japanese.pdf

[3] 総務省：“テレワークセキュリティガイドライン.”
https://www.soumu.go.jp/main_content/000545372.pdf