

# スクラムリファレンスカード

by Michael James and Luke Walter

## スクラムについて

### 経験主義フレームワーク

スクラムはクロスファンクショナルチームでプロダクト開発を行う為のフレームワークです。実際のマーケットからのフィードバックを受ける経験主義であること、そして自己管理チームである事を重要視しています。

役割、イベント、ルール、アーチファクトが提供され、チームはこのフレームワークの範囲内で自分達のプロセスを作り、適応していく責任があります。

スクラムは固定されたスプリントという1ヶ月以下（1か2週間を推奨）のイテレーション単位にエンドユーザー視点で利用可能で、出荷可能（しっかりとテストされた）なプロダクトインクリメントを作ることを求められます。

### ウォーターフォールの代替

スクラムはインクリメンタルでイテレーティブなアプローチで従来のフェイズ分けされたウォーターフォールでの開発とは異なり、価値の高いフィーチャーから実装し、早くフィードバックを得ることができます。

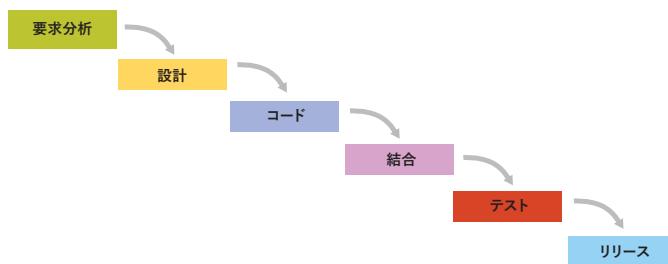


図 1: 従来のウォーターフォール開発は初期に要求を完全に理解していることと、各フェーズで問題がほぼおこらないということを前提としています。

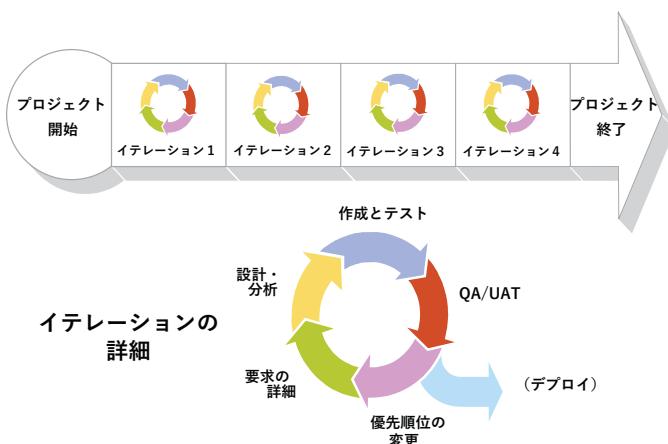


図 2:スクラムは開発に関わる全ての活動を各イテレーション内に混ぜ込み、固定されたインターバルで新たに発見した現実に適応していきます。

スクラムのメリットを多く享受できるのはプロダクト開発のような複雑な知識創造と協働が必要な環境です。多くの場合はオブジェクト指

向言語により実装されます。現在では半導体、住宅ローン、車椅子の開発にも使われるようになってきています。

### スクラムをやっているのか、やっているフリをしているのか？

スクラムはあなたに容赦なく個人、チーム、組織の課題を突きつけてきます。多くの人は、課題を突きつけられた時にスクラム自体を変化させて対応してしまい、結果的にスクラムのメリットの多くを享受できない状態になってしまいます。

## スクラムチーム

### チーム

- 「開発者」と呼ばれることがあるが、クロスファンクショナルである（機能横断性を伴う）ことが意図されており、テスト技術、ビジュアルアナリスト、デザイナー、特定分野の専門家など通常は開発者とは呼ばれない人達も含まれる。
- マネージャーが不要で自己組織・自己管理されている。
- 1スプリント単位でプロダクトオーナーや他のチームとスプリントプランニングを行う。
- どのように実装するか（How）の権限を有する。
- チーム内、外を問わず協働する（例えば、他チームとの調整、エンドユーザーに要求の明確化を行うなど）。
- 特に最初の数スプリントは1つの部屋に集まっていることが成功するるために重要である。
- 長期固定でフルタイムが成功の重要な要素である。スクラムでは学習を促進する為にチームを長期間固定し、個人ではなくチームに仕事を与える。人の移動やチームの分割は避ける。
- 6名程度（多少の増減はある）。
- リーダーが指名される事はなく、健全なチームは状況に応じて自然とリーダーシップをお互いに取り合う状態になる。

### プロダクトオーナー

- ビジョンを宣言し、優先順位を決める事で開発にかけた投資効果を最大化する。
- 複数チームであっても1つのプロダクトに1名。
- 常にプロダクトバックログの優先順位を入れ替え、リリース計画等を考慮し、調整する。
- 要求に関する意見が分かれた際などに最終的な決定権を持つ。
- リリース判断と開発継続の判断を行う。

### スクラムマスター

- スクラムを導入できる環境を組織に作っていく。
- スクラムが正しく理解されていることを担保し、導入を支援する。
- 自己組織チームを作ることのできる環境を準備する。
- チームを外部の妨害から守り、フロー状態を維持する。
- 良いエンジニアリング技法の導入を支援する。
- チームに対してマネージメントの権限を持たない。
- 課題解決の支援を行う。
- チーム、プロダクトオーナー、組織にサービスを提供する。詳しくは、<https://scrummasterchecklist.org>を参照。

# スクラム イベント

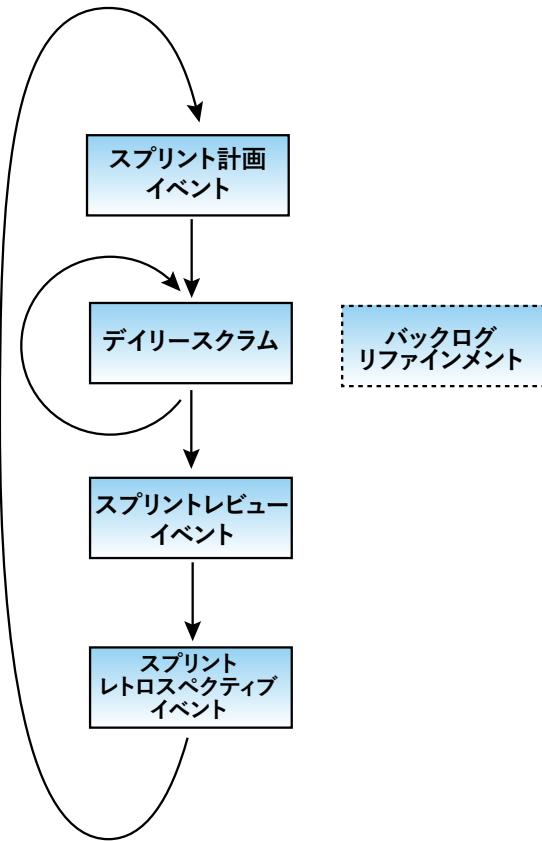


図3:スクラムの流れ

## スプリントプランニング

スプリントの初めにプロダクトオーナーとチームがスプリントプランニングと一緒に実施し、次のスプリントでどのプロダクトバックログアイテムを実装するかを検討します。プロダクトオーナーはどのアイテムがビジネス上最も重要なかを明確にします。開発チームは現状の技術的負債の状態を考慮した上で、開発できる量をプロダクトバックログから持ってきてスプリントバックログを作ります。

スプリントゴールを宣言する事でスプリントの全体像の把握と、何に集中すべきか明確になります。

チームが複雑で不明瞭なアイテムをスプリントで実装する場合は協力して、その時点で知っている情報を元に直感でどこまでできるかを予測します。時間見積りとチームの仕事に充てられる合計時間を比較し、どこまでできるかの判断を行う方法はチームが詳細なプランニングをしているフリを助長してしまい、チームからオーナーシップを奪ってしまうので、おすすめしません。この方法を使うのは最初の数スプリントに限定するか、最初からやらないことが好ましいです。

出荷可能なプロダクト単位で開発ができるようになるまでの間は、実装する機能を減らし、テスト、結合、ソース理解を進め、技術的負債が増え続ける状況を回避しないと、図14で示すような状態になってしまいます。

もし、プロダクトバックログの上方のアイテムがリファインメントされてないようであれば、計画の時間で行ってください。

スプリントプランニングが終わりに近づくと、チームはどのように仕事を終わらすか明確にしていきます。例えばアイテムをスプリント実施するタスクに分割していきます。

スプリントプランニングに使える最大の時間（タイムボックス）は30日スプリントの場合は8時間です。スプリント期間が短い場合は同じ比率でスプリントプランニングの時間も短くします。

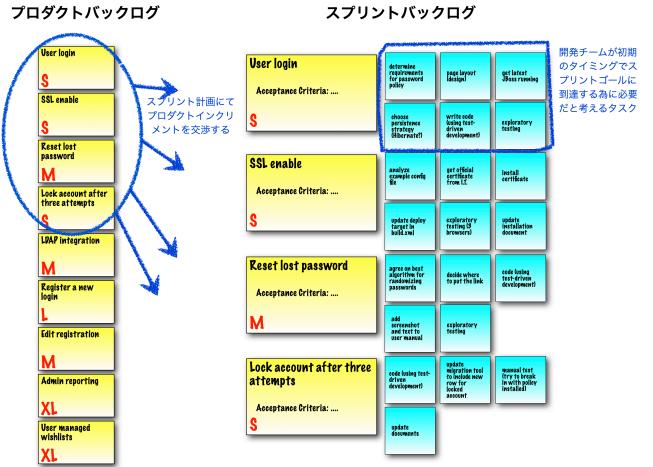


図4:スプリントプランニングのアウトプットはコミットされたプロダクトバックログアイテム(PBI)と対象アイテムを分割したスプリントのタスクとなります。

## デイリースクラムとスプリントの実施

毎日、同じ時間に同じ場所で開発チームは15分を共に過ごし、スプリントゴールに向けての進捗状況を確認し、その日の計画を行います。

デイリースクラムは多くのチームは物理的なタスクボードなどの情報発信をする物の近くに集まり、立って行う事で、効果的に時間を短くしやすくなります。引き続き議論が必要な話題が出た場合はデイリースクラム後に気になる人が残って話し合うようにしてください。

実装を進めていると、スプリント期間中にスプリントゴールを達成する為に必要な追加のタスクを発見することがよくあります。

デイリースクラムには従来のバラバラに働く習慣を減らす狙いがあります。従来の働き方に戻らないように注意が必要です。さらにチームの協働を推進するにはモブプログラミングも有効です。

スプリントの最中にチームは完成の定義を厳密に守ります。例えばですが、テストをせずに単にコードを書いただけでは出荷はできないので、完成していない事になります。未完成のアイテムはプロダクトバックログに戻し、プロダクトオーナーが優先順位を決め直すことになります。

## スプリントレビュー

スプリントレビューの目的はプロダクトインクリメントを検査し、計画を適応させる事です。プロダクトオーナーが意思決定をする上で、顧客、エンドユーザー、その他のステークホルダーから情報を得る場でもあります。

スクラムマスターはプロダクトオーナーやステークホルダーがフィードバックを元に新しいプロダクトバックログアイテムを作るのサポートする場合があります。アイテムの増加はチームが対応できるペースよりも早い場合が多いです。新たなアイテムが元々あるアイテムより重要なのであれば、優先順位が変えられますし、置き換える場合もあります。また、いくつかのアイテムは実装される事はなく捨てられます。

新しいプロダクト開発、特にソフトウェアの場合、初期に完成状態をイメージすることは難しい場合が多いです。顧客も動くソフトウェアを触って初めて自分達が本当に欲しいものを理解します。このように価値を提供しながらの反復型開発により従来の計画主導な開発では作れなかったプロダクトが作れるようになってきています。

## スプリントレトロスペクティブ（ふりかえり）

全てのスプリントはレトロスペクティブで終わります。このイベントではチームは自分たちのふるまいやプロセスを検査し、プロセスを見直し、次スプリントでの改善アクションを決めます。

専任のスクラムマスターはレトロスペクティブが退屈だったり、恐い場にならないように工夫します。深い意味のあるレトロスペクティブには心理的安全が確保されている必要がありますが、殆どの組織ではこのような安全な環境は作られていません。心理的安全性が確保されてないないと、レトロスペクティブでは不都合な問題が隠され表面化しないか、批判や敵意に満ちた場になってしまいます。

もう一つ気づきの多いレトロスペクティブになるのを妨害している要因は人間の特性でもあるのですが、結論を急ぎすぎることです。「アジャイルレトロスペクティブ」という有名な本があるのですが、その中で、ゆっくりとした意思決定にする方法がいくつか紹介されています。例えば、場を作り、データを集め、気付きを共有し、何をすべきかを決定し、レトロスペクティブを終わらせる。もう一つ、スクラムマスターにオススメなのが、「*The Art of Focused Conversations*」という本で、同じように会話の流れを「対象、思考、説明、意思決定（ORID）」というステップに分割する方法が示されています。

心理的安全の3つ目の障害は地理的な分散です。地理的に距離のあるチームは同じ部屋で仕事をしているチームほどは上手く協働できません。

レトロスペクティブでスクラムマスターには様々なファシリテーション技術が必要になってきます。例えば、サイレントライティング、タイムライン、サティスファクションヒストグラムなど。これらのツールが共通して目指している事は、複数視点を取り入れた共通認識を作り、チーム又は組織を次のレベルに進ませることです。

大規模スクラムでは、オーバーオールレトロスペクティブという場が追加され、複数チームにまたぐ課題、組織の構造や制度などに関わる問題解決を行います。

## バックログリファインメント

（テストを受けらる方へ：1チームのスクラムの場合、リファインメントはイベントではありません）

ほとんどのプロダクトバックログアイテム（PBI）は作られたタイミングでは大きすぎたり、理解がされてない為、リファインメントを必要とします。プロダクトバックログリファインメントを多くのチームは有意義と感じるようです。スプリント中に時間をとり（スプリント期間の10%程度）、先のスプリントに向けてプロダクトバックログの準備を行います。

バックログリファインメントで上方にある曖昧で大きなアイテムはビジネス的、又は技術的な懸念を考慮しながら分割され、明確化が行われます。常に全員でやるわけではなく、時にはチームの一部の人達と顧客エンドユーザーが集まり、事前にアイテムの追加や分割を行うこともあります。

リファインメントではチームがそれぞれのアイテムにかかる労力を見積り、技術的な情報をプロダクトオーナーに提供し、優先順位付けをサポートします。<sup>1</sup>

経験豊富なスクラムマスターは仕事を薄い縦断りにし、サイズは小さいがビジネス価値を持ち続けられる分割をチームが行い、しっかりと

したテストやリファクタリングがされるような完成の定義を作るのをサポートします。

プロダクトバックログアイテムはユーザーストーリーの形式で書かれることが一般的です。<sup>2</sup>このアプローチでは大きすぎるアイテムはエピックと呼ばれます。従来の開発ではフィーチャーを水平に分割します（ウォーターフォールのフェーズをイメージしてください）。水平分割の場合は依存関係が生じるので、顧客視点でビジネス価値に応じた独立した優先順位付けができません。これが習慣化されてしまうと変えるのは難しくなります。

アジャイルな状態になるには大きなエピックを小さなユーザーストーリー形式のプロダクトフィーチャーに分割できるようになっていく必要があります。例えば、医療歴を管理するアプリケーションを作る際に「医師が患者のアレルギーに関する情報を全て見られる」というエピックから「アレルギーを持っているか否かを表示する」というストーリーが作されました。アレルギーに関するデータ分析は技術的にも難易度が高かったのですが、実は医師にとって最も重要な情報は、そもそも患者がアレルギーを持っているかどうかだったのです。この例では、アレルギーの有無を表示するだけにした上で、元々のエピックが提供したかったビジネス価値の80%を20%の労力で提供することができました。これが可能になったのは、ビジネスの人とエンジニアが協力してエピックをストーリーに分割したからです。

大きなアイテムを分割する事でユーザーへの価値提供のサイクルを短くすることになり、ユーザーの本当の要求をより早く知ることにつながります。

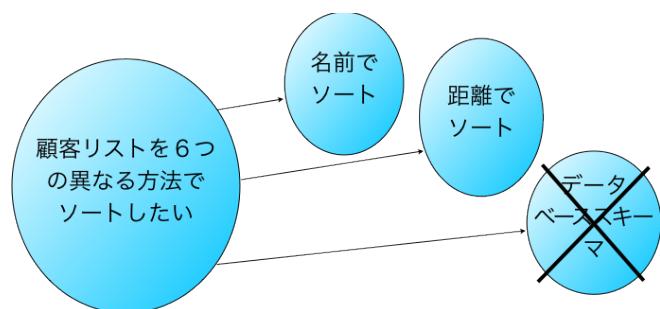


図5: バックログリファインメントでは大きなPBI(エピックと呼ばれたりします)で最上部にあるものを縦に分割していきます(ストーリーと呼ばれたりします)。開発のフェーズ単位に横切りにするのは好ましくありません。

<sup>1</sup> アイテムの見積りはチームが協働しておこないます。

<sup>2</sup> User Stories Applied: For Agile Software Development, Addison Wesley, Cohn (2004)

# スクラム アーチファクト

スクラムでは3つのアーチファクトを定義しています：プロダクトバックログ、スプリントバックログ、インクリメントです。

## プロダクトバックログ

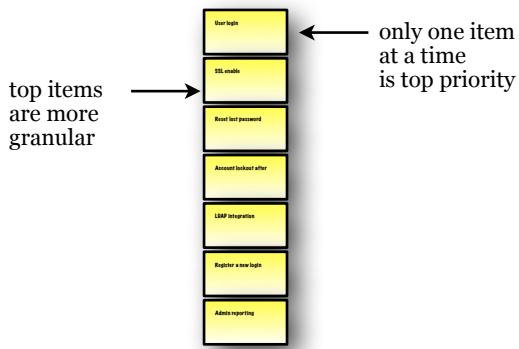


図 6: プロダクトバックログ

- 一意に優先順位付けされた作りたいもの一覧。
- 全てのステークホルダーが閲覧可能。
- ステークホルダー（チームも含む）がアイテムを編集できる。
- 常にプロダクトオーナーにより優先順位が変更されている。
- 常にスクラムチームにより更新されている。
- 上部にあるアイテムは下部にあるアイテムより小さくなっている（例：スプリントの1/4以下になっている）プロダクトバックログアイテム（PBI）。

## プロダクトバックログアイテム（PBI）

- 顧客中心のフィーチャーであり、「どうやって（How）」よりも「何を（what）」をの説明をしている。
- ユーザーストーリーのフォーマットで書かれることが多い。
- 技術的負債を避ける為にプロダクト全体に対して完成の定義が作られている。
- アイテムの内容に特化したアクセプタンスクリティアが書かれていることがある。
- 開発チームにより見積りがされており、相対見積りがされているのが好ましい（例えばストーリーポイントなど）。



図 7: 1つのPBIが顧客中心のフィーチャーを表しており、多くの場合は完成の定義を満たす為にいくつかのタスクをこなす必要がある

## スプリントバックログ

- スプリントプランニングでプロダクトオーナーとチームで協議し選ばれたPBIがあること。
- スプリントゴールの達成の障害になるのでスプリント中の変更は行わない。
- スプリントプランニングでチームにより初期のタスクが作られる。

- スプリント実行中にスプリントゴールを達成する上で必要なタスクが発見されたら追加する。
- チームにとって可視化されている状態である。
- デイリースクラムの時に参照される。

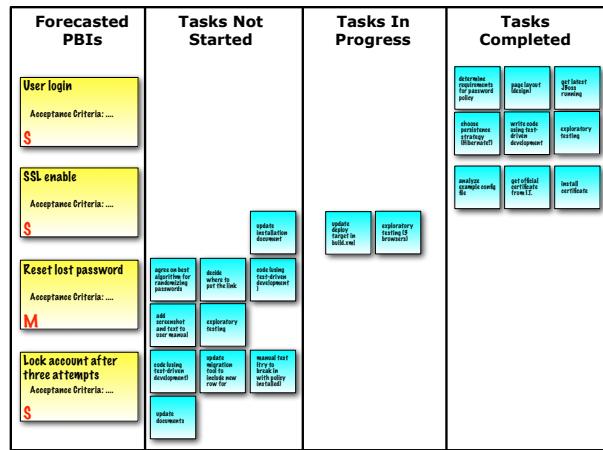


図 8: 情報の可視化の観点で、物理的なタスクボードがある方が好ましい

## （出荷可能なプロダクト）インクリメント

- スプリントでプロダクトとしての性能が完成している。
- スプリントの終わりまでに、利用可能でリリース可能な状態になっている。
- プロダクトオーナーが必要と思うタイミング、頻度でリリース可能。
- 毎スプリントレビューで検査がされている。
- 完成の定義は全てのチームが作るPBIに共通して適応される。例えば全ての変更は：
  - しっかりとテストがされている。
  - 完全に結合されている。
  - 相互にコードがレビューされている。又はペアプロやモブプロで開発がされている。
  - ドキュメントが作成されている（必要な場合）。
  - コードの可読性が維持又は改善されている。

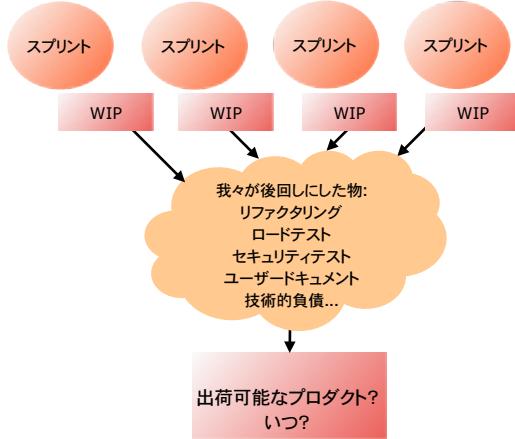


図 9: Undoneな仕事はリスクと遅延の原因になる

## スプリントタスク（任意）

- PBI(What)をどう作るか(How)が説明されている。
- 通常は1日かそれ以下で行われる。
- スプリント実行時には誰かが自主的に手をあげて、タスクのメインの責任者になる。
- チーム全員の所有物で、協働することを期待されている。

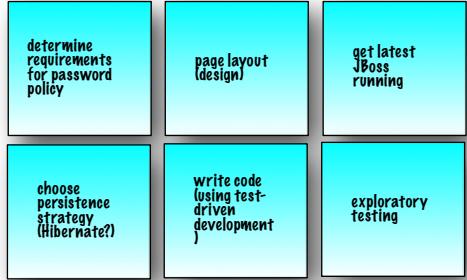


図 10: PBIを完成させる為に行う必要な全ての事がスプリントタスクになっており、従来のフェーズで切られるような状態にはなっていない（フェーズの例：要求の明確化、分析、設計、実装、デブロイ、テスト）

## 複数チーム

### あなたの組織はアジャイルな状態にならないように設計されている

スクラムは組織の複雑性を取り除く事を意図しています。組織が変化したふりをして、実情は何も変わっていない事が、大きな組織では特によくあります。大規模なスクラム導入では組織の変化は必須で、トップダウンとボトムアップの両方向からのアプローチが求められます。

スクラムは不明確な要求や技術的リスクを複数の領域に精通している人々を集め、1つのチームとし、1つの部屋に集めることで、モチベーションを高め、可視化を促進し、信頼を築くことで対応しようとしています。

人を増やしすぎることは状況を悪化させます。専門領域の単位でグループを作るのも悪い方向に向かいます。アーキテクチャ上のコンポーネント単位で人を集めてグループ（コンポーネントチームと呼ばれます）を作るとも良くない考え方です。

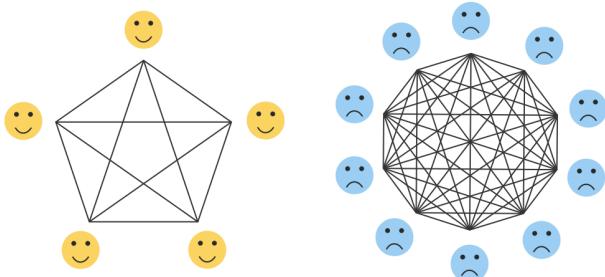


図 11: コミュニケーション経路はチーム人数に応じて指数的に増えてしまいます。

## フィーチャーチーム

完全にクロスファンクショナルな「フィーチャーチーム」は、エンドユーザーに顧客中心な機能を提供する為に必要な全てのアーキテクチャ上のコンポーネントに触れる必要があります。特に大規模なシステムでは多くの学習が求められます。チームの重要視するものが短期的な効率性ではなく学習することであれば、「学習する組織」になっていく大きな手助けとなるでしょう。

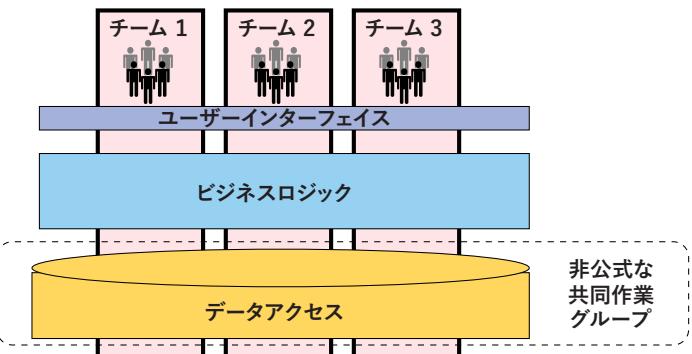


図 12: フィーチャーチームはアーキテクチャ上のコンポーネントを横断した学習を行う

### 1つのプロダクトバックログ、1人のプロダクトオーナー

大規模スクラムでは、1人のプロダクトオーナーが優先順位付けしたプロダクトバックログを複数チームで共有します。もちろんバックログのメンテナンスも協働して行います。他チームに依存した状態にならないよう、スプリントを同じリズムで実施し、このスクラムリファレンスカードに記載されたスクラムイベントのオーバーオール版および複数チーム版を、チームが決めた代表者で横断的に行います。また、複数チームも单一チームの場合と同様に、適切にテストされ、統合され、出荷可能なインクリメントを毎スプリント開発します。

# 関連する手法

## リーン

スクラムはソフトウェア開発におけるアジャイル運動のなかでうまれてきた一般的なフレームワークです。そして、リーン生産方式、特にトヨタ生産方式から多くの影響を受けています。

## エクストリームプログラミング(XP)

スクラムは技術的な手法について特に指定はありませんが、スクラムマスターは完成の定義の拡張を支援していくことに責任を持ちます。

完成の定義を満たしているアイテムは一度完了したらその後着手する事はありません。自動化されたレグレッションテストがあることで、次のバックログアイテムを着手した事により既に完了したアイテムに対する影響を防ぐことができます。知識が少ない開発初期に設計、アーキテクチャおよびインフラ等を確定せずに創発的に開発し、常に見直しながら完成度を高めています。

スクラムマスターはチームがXPなどのエンジニアリング技法を学ぶ事を促進していきます。例えば、継続的インテグレーション（継続的自動テスト）、テスト駆動開発（TDD）、継続的なリファクタリング、ペアプログラミング、頻繁なチェックインなどを取り入れていく事で、技術的な負債を防ぎます。

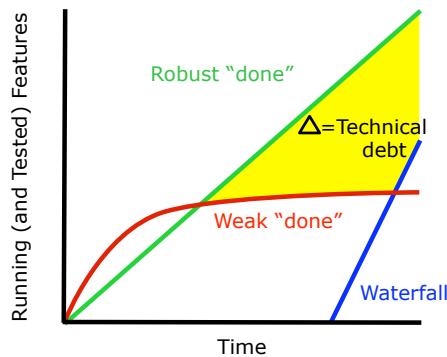


図 13: 緑の直線がアジャイル手法が目標としている持続可能な価値のあるフィーチャーを早期に提供することを表しています。完成の定義を守る能力を向上し、技術的な負債を防ぐ状態が作れると正しくスクラムを実践していると言えます。

# 自己組織化されたチーム

献身的なチームは操作されているチームのパフォーマンスを凌駕する

スプリント遂行中にチームメンバーが共有された目標に興味をもち、達成に向かってお互いに管理しあう能力を身に付けます。集団として責任を持つという考え方は多くの作業者にとって、従来の習慣に反しており、考え方を変えるには困難が伴います。管理者にとどまらず従来の罰と報酬による管理に慣れており、チームが自発的に行動する事を許容するには困難が伴います。初期段階で不快感はあるものの、スクラムマスターの観察力および説得能力が成功の確率を向上させます。

## 挑戦と機会

自己組織化されたチームは、従来型の管理されたチームの生産性を大幅に上回ります。家族と同じ位の規模のグループであれば、下記の条件をクリアすれば自然と自己管理されたチームとなっていきます。

- ・メンバーが明確な短期間の目標に献身的な状態になっている。
- ・メンバーがチームの進捗状況を把握している。
- ・メンバーがお互いの貢献度を観察している。
- ・メンバー安心して正直なフィードバックを共有できる。

心理学者のタックマン氏はチームの成長段階を「形成・混乱・統一・機能」と定義しています。最適な自己組織化に至るには時間を要します。初期段階の自己組織化チームは従来の管理型チーム生産性を下回る事もあります。

多様性の高いチームの多くは類似性が高いチームよりも優れています。初期段階では多様性により意見の対立が多く起こる事は自然なことであり、健全です。チームの生産性はこれらの対立への対処をいかにうまく行ったかで変わっていきます。

腐ったリンゴ理論によると、一人の否定的な個人（やっている仕事をグループから隠す、否定的な活動を行う又は対人関係を悪化させる振舞いをする）がグループ全体に多くの悪影響を与えてしまいます。腐ったリンゴは極少数ですが、取り除くには大きな労力を必要とします。チームが誰をチームメンバーとして受け入れるかを決められるようにする事で、このような事態を回避する可能性を高めることができます。

上司と部下の関係（高プレッシャー又はマイクロマネージされている環境）で結果を出せていない人がスクラムチームで輝くことも珍しくありません。

自己組織化は環境要因によって阻害される事があります。例として、地理的に離れている、上司部下の関係にある、パートタイム、スプリントゴールに無関係な仕事による妨害などが挙げられます。多くのチームには、このような課題の解決に尽力してくれるフルタイムのスクラムマスターが必要です。

## 著者について



マイケル ジェイムスは長年にわたりプログラミングを学習してきました。そして、ケンシュエイバーから直接学び、スクラムのトレーナーになりました。技術者、マネージャーとして経営者に価値提供にビジネスを最適化するコーチングを提供してきました。フィードバックは下記までお願い致します。

mj4scrum@gmail.com 又は <https://twitter.com/michaeldotjames>



ルーク ウォルターは工業デザイナーとして創発的プロダクト開発を長年にわたり学んできました。マイケル ジェイムスと開発チームと一緒に仕事をしたのが、スクラムとの出会いでした。その後二人ともトレーナーになりました。彼はビジネスの人々に無駄な技法に気づいてもらい、顧客価値に集中した状態を作るコーチングを提供しています。フィードバックは下記までお願い致します。

[walter.lukef@gmail.com](mailto:walter.lukef@gmail.com)

翻訳：Odd-e 榎本明仁

コーチングや研修の依頼、翻訳に関するフィードバックは下記までお願いいたします。[aki@odd-e.com](mailto:aki@odd-e.com)

## スクラムが適切な状況とは？

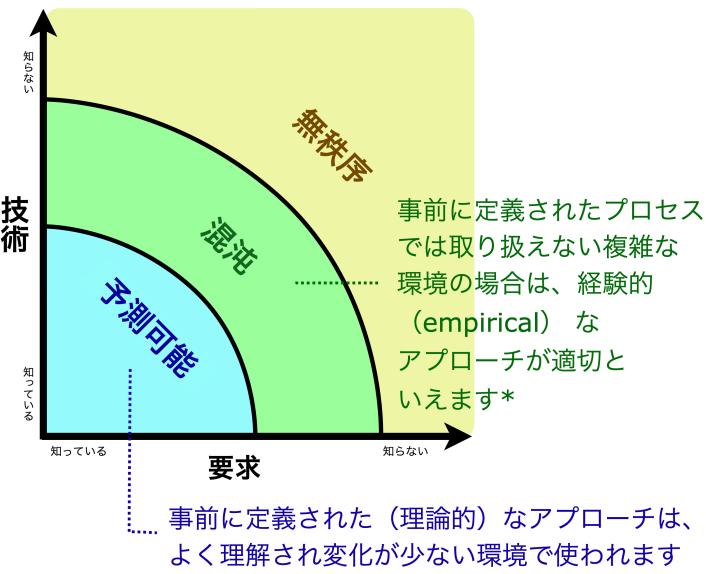


図 14: スクラムは創発的フレームワークであり、要求や技術の不確実性がある環境に適しています。

固定的なプロセスでは管理不能な仕事をどう管理していくかという課題を解決するのがスクラムの目的もあります。不明確な要求と予測不能な技術はリスクです。スクラムを導入するか、PMBOK®ガイドにあるような計画駆動開発を導入するかについて決めるには、内容をよく理解しているのか、知識を創造し協働しなければならないのかによってきます。例えばですが、スクラムは使い回しをするような開発やサービスを想定していません。また、自己組織化していくことに対してある程度の覚悟が必要です。