Scrum Starter Guide

吉羽龍太郎 (@ryuzee)　著

2019-12-01 版　　発行

はじめに

本書の目的

スクラムでは、図１のとおりプロダクトバックログが用意されていて、それを元にスクラムチームでスプリントプランニングを実施し、スプリント期間中毎日デイリースクラムを行い、最後にスプリントレビューとレトロスペクティブを実施することになっています。

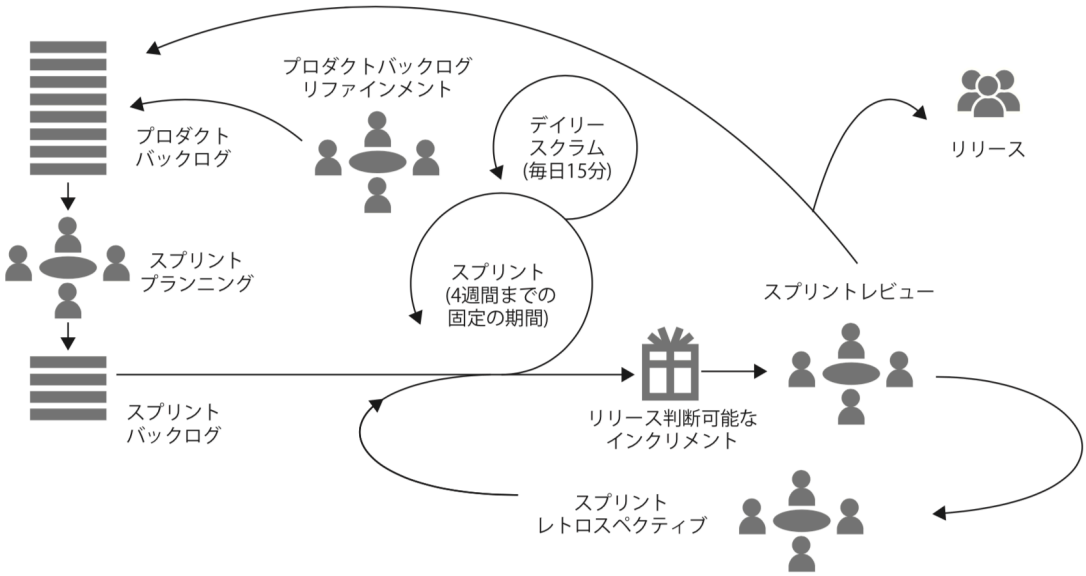


図 1: スクラムの全体像

つまりプロダクトバックとスクラムチームが存在するところがスタート地点になっています。言い換えるとそれらがないとスプリントが開始できません。本書では、実際にスクラムでスプリントを開始する前にどんな準備を行うと良いのかを考察していきます。

本書の前提

本書では、すでにスクラムのフレームワークの全体像を理解していることを前提としています。スクラムのイベント、作成物、ロールなどについて詳しく知るには、スクラムガイドを参照してください。

自分たちで考える

> 優れたプラクティスのコレクションを固定的なメソッドやフレームワークとして扱うのは、複雑系の特性を無視してしまっている。複雑系（人間を含む）は、固定と停滞から学ばない。不確実性、変動性、驚きから学ぶのだ。
> - Jurgen Appelo『Management 3.0』、『How to Change the World』などの著者

> あなたのスキルレベルによってプラクティスは異なる。他の人の経験はあなたのものと同じではない。ゆえに、単純に “カーゴ・カルト” することはできないのだ。
> - Andy Hunt『アジャイルプラクティス 達人プログラマに学ぶ現場開発者の習慣』、『リファクタリング・ウェットウェア —達人プログラマーの思考法と学習法』などの著者

カーゴ・カルト（積荷崇拝）とは、いつの日か、先祖の霊・または神が、天国から船や飛行機に文明の利器を搭載して自分達のもとに現れる、という現世利益的な信仰で、ニューギニア諸島で多く見られたものです。つまり、ある戦略が成功を導いた理由や状況を完全に理解せずに、見よう見まねで行動を真似たり、ツールを導入したりすることをカーゴ・カルトと呼びます。ですがこのようなやり方ではうまくいきません。

「〇〇というプラクティスをやっておけばよい」という思考は捨ててください。どれだけ流行っていたりメジャーなプラクティスでも同じです。つまり初期フェーズの組み立てを自分たちで考える必要があるということになります。もちろん、プロジェクトやプロダクトがうまくいかない点には類似性があり、既存の知識やプラクティスは参考にはなりますので、全てをゼロから考えなければいけないわけではありません。

立ち上げ時によく使われるプラクティスと 4 つの観点

初期の段階で行われることが多いプラクティスの 1 つにインセプションデッキがあります。アジャイル開発のプラクティスの 1 つで、どの方法論や手法にも属してない独立したものです。

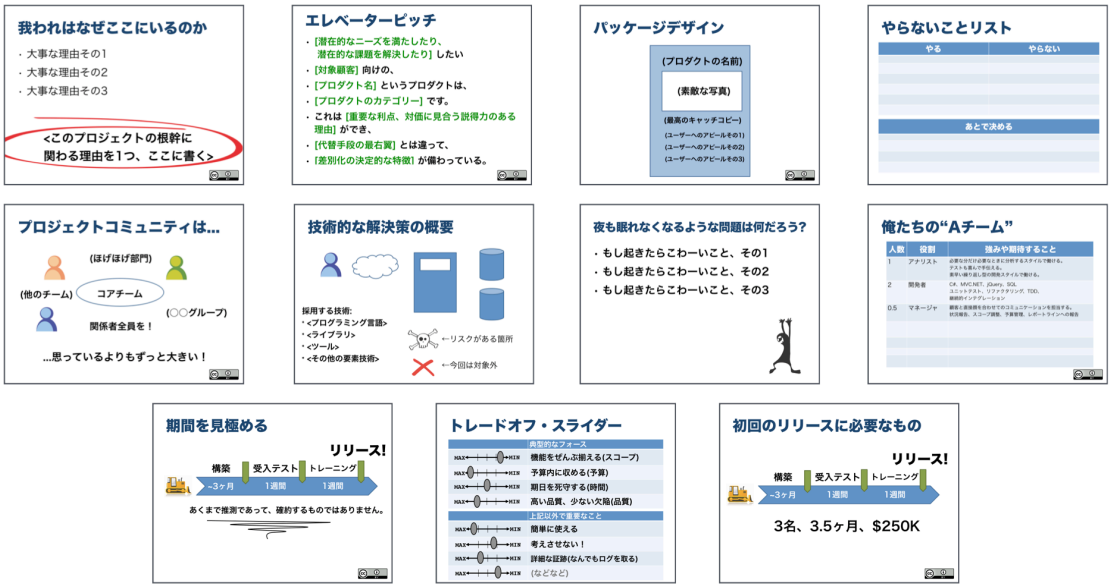


図 2: インセプションデッキ

開発を始める前に明らかにしておくべきこと 11 個の質問に答える形で整理し、チーム全員で話し合い合意したものを見える化していきます。こうしてこれから向かう先や考え方を共有していきます (憲章)。もちろん状況は刻一刻と変わるので、必要に応じて定期的に更新していきます。

インセプションデッキには図 2 のとおり次の 11 個の質問が含まれます。

- 我々はなぜここにいるのか
- エレベーターピッチ
- パッケージデザイン
- やらないことリスト
- ご近所さんを探せ
- 技術的な解決策の概要
- 夜も眠れなくなる問題は何？
- 俺たちの A チーム
- 期間を見極める
- トレードオフを決める
- 初回リリースに必要なもの

これらの質問に答えるわけですが、忘れてはいけないいちばん重要な点として、ドキュメントを作ることが目的ではないことが挙げられます。形骸化したドキュメント、使われないドキュメント、本音が隠された建前のドキュメントなどに意味はありません。インセプションデッキの目的は共通理解の形成であり、この 11 個の質問は共通理解を形成すべき領域を明らかにしていると考えれば良いでしょう。

本書ではインセプションデッキを作るのではなく、インセプションデッキに含まれる項目をそれぞれブレークダウンして考えて行きます。

インセプションデッキの 11 個の質問を共通理解を形成する観点でざっくり分類すると以下のようになります。

プロダクトに対する観点

我々はなぜここにいるのか / エレベーターピッチ / パッケージデザイン / やらないことリスト

人に対する観点

ご近所さんを探せ / 俺たちの A チーム

技術や品質に対する観点

技術的な解決策の概要 / トレードオフを決める

計画やマネジメントに対する観点

夜も眠れなくなる問題は何？ / 期間を見極める / 初回リリースに必要なもの

これらの 4 つの観点は相互に影響を与えます。本書ではそれぞれを順番に見て行きます。

著者について

吉羽 龍太郎（よしば りゅうたろう）

株式会社アトラクタ Founder 兼 CTO / アジャイルコーチ。
アジャイル開発、DevOps、クラウドコンピューティングを中心としたコンサルティングやトレーニングに従事。野村総合研究所、Amazon Web Services などを経て現職。認定チームコーチ（CTC） / 認定スクラムプロフェッショナル（CSP） / 認定スクラムマスター（CSM） / 認定スクラムプロダクトオーナー（CSPO）。 Microsoft MVP for Azure。青山学院大学非常勤講師（2017-）。著書に『SCRUM BOOT CAMP THE BOOK』（翔泳社）、『業務システム クラウド移行の定石』（日経 BP 社）など、訳書に『カンバン仕事術』（オライリー・ジャパン）、『ジョイ・インク』（翔泳社）など多数。
Twitter：@ryuzee ブログ：https://www.ryuzee.com/

著作について

本書のほかに執筆、翻訳した書籍のうちお勧めのものについて紹介します。

- レガシーコードからの脱却 ―ソフトウェアの寿命を延ばし価値を高める 9 つのプラクティス
- Effective DevOps ―4 本柱による持続可能な組織文化の育て方
- 変革の軌跡【世界で戦える会社に変わる"アジャイル・DevOps"導入の原則】
- ジョイ・インク 役職も部署もない全員主役のマネジメント
- アジャイルコーチの道具箱 - 見える化の実例集
- カンバン仕事術 ―チームではじめる見える化と改善
- SCRUM BOOT CAMP THE BOOK

ライセンス

本書記載の内容は引用部分を除き著者が著作権およびその他隣接する権利を有しています。本書の一部または全部の複製、改変、頒布はできません。

本書は著者の経験などをまとめたものであり、著者の文脈で機能したやり方の 1 つとして紹介するものです。すべての状況において当てはまるものでもなく、本書を利用した以下なる結果についても著者は責任を持ちかねますので予めご了承ください。

本書に関するご意見、お問い合わせ

本書に関するご意見、お問い合わせは、Twitter：@ryuzee までお願いします。

目次

はじめに **ii**
本書の目的 ii
本書の前提 iii
自分たちで考える iii
立ち上げ時によく使われるプラクティスと 4 つの観点 iii
著者について v
著作について v
ライセンス vi
本書に関するご意見、お問い合わせ vi

第 1 章 プロダクトの観点 **1**
1.1 プロダクトゴールや価値の明確化 1
1.2 クネビンフレームワークによる問題の分類 2
1.3 スクラムによる開発の合意 3
1.4 初期のプロダクトバックログの準備 3

第 2 章 人の観点 **5**
2.1 ロールの明確化 5
2.2 プロダクトオーナーの決定と教育 5
2.2.1 誰がプロダクトオーナーをやるべきか 7
2.3 開発チームの適切なメンバーの選定 8
2.4 スキルの見える化 9
2.5 スクラムの知識レベルを揃える 10
2.6 ファシリティ 10
2.7 道具を揃える 10
2.8 ワーキングアグリーメントを設定する 11

第 3 章 技術や品質の観点 **13**
3.1 非機能要件の明確化 13
3.2 アーキテクチャや開発言語などの選定 14

3.3 完成を定義する 14
3.4 ドキュメントの扱い 15
3.5 スプリントにおける活動 16
3.6 開発環境の作成 16
3.7 テスト自動化の準備 17
3.8 プラクティスの選択 17

第 4 章 計画やマネジメントの観点 19
4.1 見積りに影響するパラメータ 19
4.1.1 チームのサイズが決まっている場合の費用の見積もり 19
4.1.2 期間を優先し、チームのサイズをあとから決める場合 20
4.2 スプリントの長さを決める 20
4.2.1 スプリントは短いほうがよい 21
4.2.2 1 週間スプリントの弊害 24
4.3 リスクを管理する 25
4.4 報告 25
4.4.1 メトリクスの間違った活用に気を付ける 25

第 5 章 初期フェーズの進め方 27

付録 A スクラム関連用語集 28
A.1 基本用語 28
プロダクトオーナー 28
開発チーム 28
スクラムマスター 28
スクラムチーム 29
ステークホルダー 29
スプリント 29
スプリントプランニング 29
デイリースクラム 29
スプリントレビュー 30
スプリントレトロスペクティブ 30
プロダクトバックログリファインメント 30
プロダクトバックログ 30
スプリントバックログ 30
インクリメント 31
A.2 関連用語 31
完成の定義 31
スプリントバーンダウンチャート 31

スクラムボード 31
相対見積り 31
受け入れ基準 （Acceptance Criteria） 32
スプリント 0 32
ベロシティ 32
ユーザーストーリー 32
妨害事項リスト（Impediments List） 32
Scrum of Scrum 33
XP（eXtreme Programming） 33

第 1 章

プロダクトの観点

1.1　プロダクトゴールや価値の明確化

初期の段階で大事なのは、これから作るもののビジネス価値やプロダクトビジョンを明確にすることです。そしてこのビジョンは、これから作るものの責任を負っている人が誰なのかが明らかでないと決まりません。つまり人の観点に依存していると言えます。

なお、この時点では詳細な要求に踏み込まず、全体像を把握することが最優先です。

また開発以前の話としてビジネスモデルの整理をしないといけないこともあります。ビジネスモデルの整理には、例えばビジネスモデルキャンバスやリーンキャンバス、ユーザーインタビューなどのツールが利用可能です。実際の開発には多額のコストがかかります。アイデアやビジネスモデルの検討が不十分で見切り発車してしまうと、多くの時間とお金をかけたあとにニーズがないことが分かった場合に大きな損害になってしまいます。もちろん必ずプロダクトが当たるという保証などできませんが、あまい見通しだけで進めるのは無謀です（逆に半年も 1 年も検討だけしていても意味はありません）。

作るものの特性が分からないとどんな開発の仕方をするかは決まりません。つまり開発手法はこの時点ではスクラム一択ではありません。では、どのようにして開発の手法を選択すればよいでしょうか。

1.2　クネビンフレームワークによる問題の分類

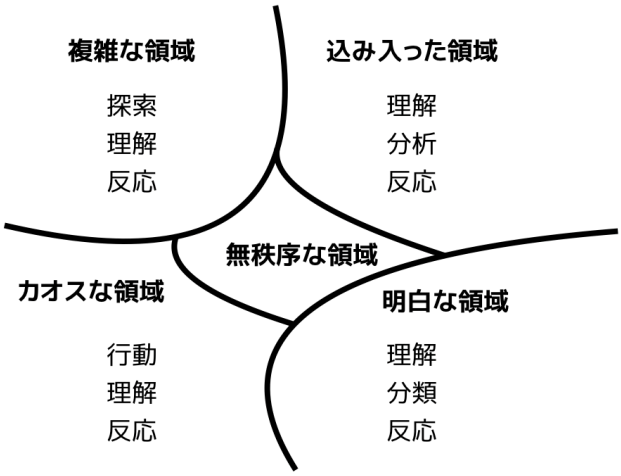


図 1.1: クネビンフレームワーク

問題の性質を分類するフレームワークの 1 つにスノーデン氏が作成したクネビンフレームワークがあります（図 1.1）。このフレームワークでは問題は 5 つの領域のいずれかに属するものと考えています。

1 つ目の明白な領域とは、正解が存在し変化の少ない領域を指します。この領域では状況を理解・分類して、過去から蓄積したベストプラクティスに基づいて進めていくのが良いでしょう。つまり過去の成功例をそのまま当てはめて進めれば良いので、ウォーターフォール型の開発と親和性が高いとも言えます。

2 つ目の込み入った領域とは、正解が 1 つではないものの、比較的予測しやすい領域を指します。状況を専門家が分析して理解し、取り得る手を検討して進めていきます。専門家の分析によって問題がわかれば、それを踏まえて過去の経験などの中からうまくやる方法を探して、それをグッドプラクティスとして活用していきます。つまり最初に分析を行い詳細が明らかになれば、そこに対してウォーターフォールが適用可能になります。

3 つ目の複雑な領域とは、問題を把握するために試行錯誤（探索）を行い、それによって状況を把握し、そのあとに次の対応を考えていく領域のことです。この場合、うまくやる方法は最初からは分からず後からわかることになります。つまり計画主導型のやり方では対応が難しく、アジャイル開発が向いている領域となります。

4 つ目のカオスな領域とは、革新的なことが求められる領域で正しい答えもわからないので、まず行動してから理解していくしかない領域のことです。この領域では既存の知識が役に立たないこともあります。つまり計画を事前に十分にたてることはできません。

これから解決しようとしている問題が明白な領域や込み入った領域であれば、従来のウォーターフォール型の開発が効率的でしょう。一方で複雑な領域とカオスな領域ではアジャイル開発が適しています。問題にあった開発手法を選択する必要があるのです。

1.3 スクラムによる開発の合意

対象プロダクトの特性を踏まえてスクラムがよいのであれば、その合意をステークホルダーから取りましょう。必要に応じて関係者にはアジャイル開発やスクラムの価値観、従来型との違いを説明して理解してもらう必要があります。アジャイル開発では、従来のプロジェクトマネジメントとは考え方が大きく異なるので、合意が取れていないと後で従来型とのギャップに苦しむことになります。また、従来の報告の仕方とは異なる可能性が高いので、どのような報告が必要かを確認し、あわないものは交渉してやめるといった準備も必要です。

よくあるギャップには次のようなものがあります。

- スクラムチームは変化に対応してスコープを柔軟に変更しながらプロダクトを作ると考えているが、ステークホルダーは最初に洗い出したスコープをすべて作ると考えている
- スクラムチームはフィードバックを選別して意味のあるものだけを取り入れればよいと考えているが、ステークホルダーはすべてのフィードバックを取り入れるべきだと考えている
- スクラムチームは開発チームの稼働できる時間（キャパシティ）や期間を固定して、その範囲で全力を尽くすと考えているが、ステークホルダーは人や労働時間を増やしても構わないと考えている
- スクラムチームは必ずしもすべてのバグを修正しなければいけないとは考えていないが、ステークホルダーはすべてのバグは重大度に関係なく修正しなければいけないと考えている
- スクラムチームはドキュメントは必要な分だけを作ればよいと考えているが、ステークホルダーはウォーターフォールで作るようなドキュメントをすべて作らなければいけないと考えている

これらはどちらかが悪いという話ではありません。いままでやってきたやり方にも、そのコンテキストでの合理性があって結果が出ていたのですから、一概に否定するものでもありません。あくまで考え方の違いです。ただし、ギャップを解消しないままで進めてしまうと、いずれも後になって問題を引き起こすので、開始前に期待値をすり合わせておきましょう。

1.4 初期のプロダクトバックログの準備

プロダクトのビジョンが明らかになり、アジャイル開発が向いているということになれば、それにあわせた要求の整理を進めましょう。このタイミングで大事なのは、まずは全体像を見ることです。細部に入りすぎるときりがないのでユーザーストーリーマッピング（図 1.2）のようなツールを使って全体の把握に務めます。見える化することによって共通理解を形成するのが目的です。

図 1.2: ユーザーストーリーマッピング

多くの場合、初期に洗い出したプロダクトバックログ項目をすべて開発することはありません。さまざまなことが明らかになっていくにつれて、不要な項目は削除され、新たな項目が追加されていきます。したがって、初期の段階ですべてが揃っている必要はありません。ただし、優先順位が高い項目は洗い出しておかないと、そもそも開発自体を始められません。ユーザーストーリーマッピングなどを使って抽出したプロダクトバックログ項目の候補を着手する順番に並び替えましょう。このときプロダクトバックログの上位の項目ほど具体的にしておきます。下位の項目はやらない可能性も高いので詳細化しすぎないようにします。

そうして項目が洗い出せたら、それぞれの項目をラフに見積もっておきましょう。ストーリーポイントを使っても構いませんし、T シャツサイズ（S / M / L / XL など）を使っても構いません。この時点の見積りはあくまで全体の把握のためなので、あまり時間をかけずに済ませましょう。並べ終わった上位の項目は、スプリント 1 に向けて受け入れ基準を用意することになりますが、これについては別途解説します。

第 2 章

人の観点

次に人の観点を見て行きましょう。

2.1　ロールの明確化

スクラムでは、プロダクトオーナー、開発チーム、スクラムマスターという 3 つのロールが定められています。この 3 つのロールをあわせてスクラムチームと呼びます。またチームの外側には、ステークホルダーと呼ばれる関係者がいるはずです。たとえば、顧客、ユーザー、スポンサー、営業、マネジメントチーム、スペシャリストチーム、QA チーム、運用チーム、社内管理部門、システム連携先などはステークホルダーの候補です。

まずはここから先を進めていく上で、誰がどのようなロールを担うのか明らかにしておきましょう。スクラムチームだけに限らず、ステークホルダーが今回のプロダクトにどのように関係していて、どんな利害があるのかも明らかにしておくと良いでしょう。

スクラムでは、スクラムチームとその外側に境界を設定します。境界を設定することで、開発チームは集中して成果を出すことに取り組めるようになります。外部との接点は、プロダクト面ではプロダクトオーナーが担い、プロセス面ではスクラムマスターが担うのが一般的です。つまり、プロダクトオーナーとスクラムマスターは外部と協働するのも仕事のうち、ということです。

2.2　プロダクトオーナーの決定と教育

スクラムで成果を出す上で、プロダクトオーナーはいちばん重要です。プロダクトオーナーは、プロダクトの責任者（結果責任）を取る舵取り役で、プロダクトの価値を最大化する責任を持つ 1 人の人（委員会ではありません。サポート役はつけても構いませんが最終決定権は 1 人に集約します）です。次のような責任を担います。

- プロダクトの結果責任を取り、ビジネス価値を最大化する責任
- プロダクトのビジョンを周りに示し理解させる責任
- プロダクトバックログの管理と並び順の最終決定の責任
- 開発チームをうまく使う責任

- 開発チームの成果物の受け入れ可否を決める責任
- スクラムイベントに参加して開発チームやステークホルダーと協働する責任
- ステークホルダーをマネージする責任
- 予算の管理責任
- リリース日を決める責任
- スプリントのキャンセル判断をする責任

なお、開発チームのメンバーにタスクをアサインしたりやり方を指定する責任や権限はありません。

プロダクトに唯一の舵取り役なので、成果を出せるかどうかはプロダクトオーナー次第です。ステークホルダーとのやりとり、開発チームとのやりとり、プロダクトバックログ項目の優先順位付け、開発チームが作ったものの受け入れ判断など、プロダクトオーナーにはやるべきことがたくさんあります。その中でも、いちばん重要な仕事が意思決定です。

すべてのものは有限です。プロダクトの開発の予算や期間は限られています。開発チームの人数も限られていますし、その一方でやりたいことはたくさんあります。すべてをやることは絶対にできません。つまり、取捨選択の意思決定ができなければいけません。「No」を言わなければいけないのです。これができる人をプロダクトオーナーとします。ステークホルダーの言うままにすべてを行わなければいけない状況であれば、その人はプロダクトオーナーとして適任ではありません。適切な人を選び直すか、権限移譲が必要です。

プロダクトオーナーは色々な悩みや問題は尽きることはありません。しかし、それを 1 人で抱え込む必要はありません。すべてのことを 1 人でやりきれるわけでもありません。困ったことがあれば、開発チームやスクラムマスター、ステークホルダーなどに助けを求めればよいということを理解する必要があります。

プロダクトオーナーが、いつどのようなことをやるか一例を挙げておきます。

初期

事業計画を考える / プロダクトビジョンを考える / マーケットの状況を理解する / 事業やプロダクトの承認を得る / ステークホルダーを特定する / 必要な権限を移譲してもらう / 予算を確保する / マイルストーンを考える / 非機能要件を考える / 初期のプロダクトバックログの作成をリードする

開発中

プロダクトバックログを並べ替える / 開発チームの疑問に答える / スプリントプランニングに参加する / 開発チームの作成物の受け入れ判定をする / スプリントレビューを主催する / ステークホルダーをスプリントレビューに招待する / フィードバックを受けてプロダクトバックログを更新する / ステークホルダーや顧客の満足度を定期的に確認する / スプリントレトロスペクティブに参加する / 全体の進捗や今後の見通しを明らかにする

終盤

リリース計画を考える / マーケティングやサポートなど他のステークホルダーと連携する / 社内プロセスの手配をする / クロージングに必要なプロダクトバックログ項目をとりまとめる

2.2.1 誰がプロダクトオーナーをやるべきか

事業責任者がプロダクトオーナーを担う

プロダクトを開発するということは、そこに事業があります。プロダクト自体は、事業を達成するための 1 要素ということになり、ほかにもマーケティングやサポート、営業などさまざまな要素が登場します。 事業責任者がプロダクトオーナーを担うということは、大きい会社であれば、たとえば事業部長が担うようなものです。スタートアップであれば創業者が該当することになるでしょう。

メリット

予算を持っており、意思決定権限が強いので、プロダクトに対する判断を素早くできる。たとえば予算やチームメンバーを追加したり、リリーススケジュールを変更したりするということも比較的容易

デメリット

プロダクト以外の仕事も多数持っていることが多いので、継続的にスクラムプロセスに関与し続けること自体が難しいことが多い。たとえばイベントにプロダクトオーナーが来れず日程調整をするようなことが増え、プロセスのリズムが悪くなったり、プロダクトバックログの手入れが十分に行き届かず見切り発車が増える。ほかの人に比べて立場が上であることが多く、開発チームとの間で健全な交渉が行われなかったり、フィードバックループが弱くなりやすい。組織のサイズが小さいうちしか機能しない可能性が高い。

ビジネスサイドの人がプロダクトオーナーを担う

たとえば、企画担当の人がプロダクトオーナーを担う例です。ある一定規模の組織になると、最初に出た事業のアイデアを具現化していくのに企画担当の人が参画して、詳細化していくことがよくあります。その上で投資判断などを経てゴーサインがかかります。このとき、引き続きその人がプロダクトオーナーになるイメージです。

メリット

ビジネスのことが分かっている（ドメイン知識がある）ので、どの順番に何をつくると良いかの判断がしやすい（意思決定の権限を事業責任者から適切に移譲されていることが前提になる）。

デメリット

技術的な知識がないことがあるため、性能やセキュリティ、運用といった非機能要件に関するプロダクトバックログ項目を用意できなかったり重要性の判断を間違えることがある（そのためスクラムマスターや開発チームがプロダクトバックログの管理を支援するのに時間を使う必要がある）。組織として企画は平行して複数進めるようになっている場合、プロダクトオーナーが忙しくなりやすい。プロダクトが生み出す成果に焦点をあてるといちばん成果を出しやすい。

UX デザイナーやアーキテクトの人がプロダクトオーナーを担う

企画が通ったあとに、それを UX デザイナーやアーキテクトなどエンジニアリングに近い人に引き渡して、進めていくやり方です。組織構造上、企画部門と開発部門が分かれているような場合によく見られる形態です。

メリット

全体の使い勝手や、プロダクトにあった品質やアーキテクチャの構築という観点が考慮された状態でバランスを取って開発を進められる。開発チームと近いスキルセットなので、開発チームとのコミュニケーションは円滑に進みやすい

デメリット

プロダクトのビジョンを開発チームに伝え続けるのが大変。プロダクトバックログの並び順や、機能をどの深さまで作るのかという点をプロダクトオーナーだけで判断することができず、企画側などのステークホルダーと常時交渉と調整が必要になる。関係性によっては常時プレッシャーをかけられることもある。時に、作りすぎてしまう可能性がある

顧客がプロダクトオーナーを担う

受託開発の文脈で検討することのある形態で、発注者側がプロダクトオーナーを担当し、スクラムマスターと開発チームを受注者側が担当する、というような形です。現実的には厳しいやり方です。

メリット

顧客が自分自身の責任でドライブできる

デメリット

いままでのやり方と関与の度合いが変わってくるが、その時間を捻出できず、開発チームに十分な時間が使えない。発注側のマインドセットのままで、対立構造となってフィードバックループが形成されない。プロセスを無視して、要求を押し付けてしまいやすい。開発チーム側がプロダクトの価値を理解できなかったり、プロダクトバックログ項目ごとの意図が分からないままに、開発しなければいけないようなことが起こりやすい。

2.3 開発チームの適切なメンバーの選定

プロダクトの開発を進めることが決まった時点で開発チームのメンバーも決まっていることも多いかもしれませんが、これから新しく開発チームを作る場合には、どのようなメンバーを集めれば良いか見ておきましょう。

適任者の特性は次のとおりです。

- 新しいことに抵抗がない
- 常に変化する状況にストレスを感じない

- 個人の責任範囲ではなく、全体の成果を気にすることができる
- 自分のことばかりではなく、人が困っているときに助けられる
- 意見や反論を言える
- コードが書ける・インフラが作れる・必要な技術スキルがある

これらはスクラムに限らずアジャイル開発全般にあてはまります。アジャイル開発は変化への対応を重視しています。したがって変化することが自然だと考えるようなマインドセットは重要です。また個人ではなくチームによるコラボレーションやコミュニケーションを重視するやり方なので、周りから信頼を得られるようなふるまいができることが求められます。もちろん、実際に役にたつものをつくる上で技術力も必要です。すべてを高いレベルで満たすことはなかなかできませんが、開発チーム全体としてこれらが実現できるようにしましょう。また、開発チームの強さはさまざまな角度から物事を考えられることなので、同じキャリアや考え方に揃い過ぎないように多様性に配慮すると良いでしょう。

開発者は必ず専任でアサインします。他のプロダクトやプロジェクトを兼任していると、それに引きづられて十分にコミットできなくなってしまったり、ボトルネックになってしまい、かえって開発チームの迷惑になることもあります。開発者が地理的に分散している場合は 1 箇所に集めて全員同席になることが理想です。もちろん働き方の多様性も重要ですが、一緒に同じ場所で働いたことのない人たちが、同じ目的をもったチームとして働くことは難しいので、1 か月程度でもよいので一緒に同じ場所で働くようにすることをお勧めします。

なお、アジャイル開発の考え方があわない人たちでチームを作ってしまうと失敗する可能性が高くなります。例えば、以下のようなケースではリスクが高くなります。

- 開発チーム内の開発者のパートナー比率が非常に高い
- 既存のチームが強力な指示型で運営されている
- 開発経験が非常に少ないメンバーだけで構成されている

2.4 スキルの見える化

開発チームができあがったら、開発チームのメンバーのスキルセットを明らかにしておくのは良い手です。見える化のためによく使われるのがスキルマップです（図 2.1）。そうすることで、成果を上げやすい技術を採用できるとともに、メンバー間でのスキルアップを進めやすくなります。

	Ruby	RSpec	HTML5	Vue.js	Fluent d	Elastic Search	Redis	AWS	Terrafo rm	CircleC I	Git	設計	運用	ファシリテーション
阿部	○	◎			・		・	△	△	△	◎	★	◎	◎
坂本	◎	○	△	△				◎	◎	△	◎	○	○	△
丸	★					◎	★	◎			◎	○	◎	△
亀井	・		◎	◎							△			
若林	・	・			◎	◎	◎		○	○	◎	○	◎	
増田	△	△		△				○		・	◎			

★:エース / ◎:得意 / ○:1人でできる / △:助けがあればできる / 空欄:できない / ・:今後習得したい

図 2.1: スキルマップによる見える化

2.5　スクラムの知識レベルを揃える

また、体制がある程度見えてきたら、さまざまな活動に本格的に入っていく前に、スクラムチーム（プロダクトオーナー、開発チーム、スクラムマスター）とステークホルダーの知識レベルを揃えておきましょう。全員を集めて、アジャイル開発の価値観、仕事の流れ、イベント、作成物、ロールなどの基本的な概念を理解して、言葉があうようにします。こうすることで、これから行う活動で、従来型の考え方をそのまま持ち込んでしまってムダが発生するのを防ぎます。またこのような活動は、その後も定期的に行うと良いでしょう。全員がプロセスに対して共通理解を持っていることは非常に重要です。

2.6　ファシリティ

これからアジャイル開発を始めるにあたっては、スクラムチーム全員が集まれる場所を用意するようにしましょう。部署が違っていても同じプロダクトを開発しているのであれば、同じ場所に集まるのです。このようなやり方を全員同席と呼びます。全員同席はエクストリームプログラミング（XP）のプラクティスの 1 つで、コミュニケーションのオーバーヘッドを大きく削減できます。専用の部屋が用意できると心理的にも安全性を確保しやすくなります。逆に、社内で初めてのアジャイル開発を行うようなケースで、いままでの環境が日常的に静かな環境の場合、他のチームからクレームがつく場合があります。こうなってしまうと、コミュニケーションが阻害されて、成果が出にくくなってしまいます。

2.7　道具を揃える

専用の場所が用意できたら、ほかに必要なものをまとめて準備しておきましょう。よく使うのは次のようなものです。

- 付箋紙

- 模造紙
- ホワイトボード
- 大きなディスプレイ
- コーヒー
- おやつ
- 冷蔵庫
- ペン、マーカー、定規、その他文房具類
- もちろん速い PC

多くの場合、いちばんコストがかかるのは人件費です。つまり人が安心して働きやすい環境を作ることは、いちばん高い「リソース」を効果的に活用する上で絶大な効果があります。

2.8 ワーキングアグリーメントを設定する

ワーキングアグリーメントとは、スクラムチームが規律を守って生産的に物事を進めるために用意するもので、仕事の働き方のルールを示したものです。Google などは良いチームのポイントとして「集団規範」を挙げていますが、このような規範を明確にするものです。ワーキングアグリーメントは、スクラムチーム全員で議論して納得した内容を明文化するものであって、プロダクトオーナーやステークホルダーが強制するものではありません。

スプリント 1 の開始前までに用意して、以後レトロスペクティブごとに見直していくと良いでしょう。守るのに注意が必要な項目に絞り込む（ホワイトボードの脇に貼れるくらいの量）、当たり前にできるようになったことを削除するなどして、軽量に保つようにします。

働き方の合意なので、スクラムチームによって内容は異なります。例を見てみましょう。

- 午前 9 時〜午後 4 時がコアタイム。コアタイム外にスクラムイベントを設定しない
- スクラムイベントは休暇以外は全員参加。不参加の際は意思決定を委ねる
- デイリースクラムは午前 10 時に開始
- デイリースクラムの前までにタスクの残時間を更新しておく
- 毎週火曜日の午前 11 時からスプリントレビューを実施する
- レビューの準備には 1 時間以内
- イベントやミーティングは時間通りに開始して、タイムボックス内で終了する
- イベントやミーティング中は携帯に出ない
- レトロスペクティブでは、スクラムマスター以外はノート PC を閉じる
- プロダクトオーナーは午前 11 時〜午後 3 時まではチームの部屋にいる
- テストがすべて終わったものを完成とする
- ビルドが失敗した場合、修正コード以外のプッシュをしない
- 誰かに助けを求められたら「よろこんで」と返事する
- 予定休暇は 1 週間前までに共有する
- マスターブランチへの直接プッシュは禁止

- ペア作業をする場合は 50 分に 1 回 10 分休憩を入れる
- ドキュメントはマークダウン形式でバージョン管理システムに入れる
- 1 つのことに 30 分ハマったら周りに助けを求める
- 体調が悪い場合は無理して来ない。周りに移す方が迷惑
- 休日の前日午後 4 時以降はリリースしない

第 3 章

技術や品質の観点

ウォーターフォールかアジャイルかに関係なく、技術や品質の観点は重要です。

プロダクトやシステムが達成すべき品質は、開発手法によって決まるのではなく、そのプロダクト自体が、誰のためにどう使われるかによって決まります。たとえば、銀行のオンラインシステムであれば、金額を間違うことも、振込が遅延することも、振込の画面でエラーがでることも許容されず、詳細なテストケースをすべて網羅しなければいけないかもしれませんが、使い捨てのキャンペーン申し込みサイトであれば、多少の不具合があっても関係ないこともあります。

つまり、どの程度の品質が必要になるかによって、開発にかかる期間やコストは大きく変わってしまいます。

3.1 非機能要件の明確化

そこで開発を始める前に行うのが非機能要件の明確化です。以下にあげるような項目 (※) について、全員が共通理解を持てるようにします。

- 可用性（継続性、耐障害性、災害対策、回復性など）
- 性能・拡張性（業務処理量、性能目標値、リソース拡張性、性能品質保証など）
- 運用・保守性（運用時間、バックアップ、運用監視、計画停止、予防保守など）
- 移行性（移行時期、移行方式、移行対象データ、移行計画など）
- セキュリティ（制約条件、セキュリティ診断、アクセス制限、データ秘匿など）
- システム環境・エコロジー（システム特性、適合規格、環境条件など）

なお、小規模な新規プロダクトの場合などでは、上記で触れた項目をすべて考えるのは無駄になる可能性があります。プロダクトがマーケットに受け入れられるか分かっていないうちに、プロダクト以外のことに多くの時間を使うのは得策ではないので、プロダクトの成長を見ながら随時必要なタイミングで検討していけば良いでしょう。

※ IPA 公開の非機能要求グレードを元に作成

3.2 アーキテクチャや開発言語などの選定

非機能要件を踏まえて、アーキテクチャや開発言語の選定を行います。たとえば以下のような項目を検討すると良いでしょう。

- プロダクトのアーキテクチャをどのようにするか
- どの開発言語でどんなフレームワークを使うか
- テスト自動化にはどんなライブラリを使うのか
- インフラに何を使うか

もちろん初期の時点では決めきれず評価や検証が必要な場合もあります。すべての検証を事前に行う必要はありません。後から変更すると大きなコストがかかるようなものについては、優先順位を上げて早めに検証しておくと良いでしょう。検証の際は、タイムボックスを設定して、時間がかかりすぎないようにします。検証の結果が芳しくない場合は、早めに変更の判断をします。ほかに検証したい項目があれば、プロダクトバックログの中に入れておくようにします。

3.3 完成を定義する

完成の定義とは、プロダクトバックログ項目が完成したと言えるために満たさなければいけない品質条件です。例えば、ユニットテストがある、統合テストが実施済である、静的解析で重大な警告がない、リリースノートが用意されている、コードがレビューされている、といったあるべき状態を定義します。プロダクトバックログ項目は、この完成の定義を満たしてはじめて「完成」となります。定義を満たしていない場合は、プロダクトオーナーの受け入れ確認をしてはいけないですし、条件付きで完成とするようなこともできません。

類似のものとして、プロダクトバックログ項目の受け入れ基準がありますが、受け入れ基準は、そのプロダクトバックログ項目が持つべき機能性の確認であり、プロダクトバックログ項目ごとに異なるのに対して、完成の定義は、複数のプロダクトバックログ項目で共通して満たす必要がある非機能性の確認となります。

完成の定義はプロダクトの性質やビジネスのゴールによって変わります。金融系のアプリケーションと使い捨てのキャンペーンサイトでは満たすべき非機能性に違いがあるのは当然です。

完成の定義は、プロダクトオーナー（最終決定者）と開発チームが中心となって考えます。プロダクトオーナーはもちろんステークホルダーと協働していく必要があります。ここでのステークホルダーには、大きな組織であれば、社内の品質管理部門などが含まれることがあるでしょう。

完成の定義を決める上では、「なぜ？」「なにを？」「誰のために？」をはっきりさせることが重要です。

プロダクトバックログ項目の完成の定義	スプリントの完成の定義	内部リリースの完成の定義	正式リリースの完成の定義
・項目がJIRAに存在する ・テストコード、プロダクションコード含め全てのコードがチェックインされている ・コードレビューを受けている ・コードフォーマッタがかかっている ・静的解析で必須修正事項がない ・全てのユニットテストをパスしている ・全ての受け入れテストが用意され、それにパスしている ・(略)	・全てのプロダクトバックログ項目の完成の定義を満たす ・プロダクトのバックアップが更新されている ・パフォーマンステストが完了している ・パッケージ、クラス、アーキテクチャのダイアグラムを更新 ・全てのバグが解決しているか、対応の時期を決めている ・ユニットテストによる全コードカバレージが80%以上 ・(略)	・JIRAでマイルストンが作成されている ・全てのスプリントの完成の定義を満たす ・インストーラーが作られている ・マニュアルが更新されている ・トラブルシューティングガイドが更新されている ・障害発生時の復旧計画が更新されている ・全てのテストスイートにパスしている ・(略)	・JIRAでマイルストンが作成されている ・全ての内部リリースの完成の定義を満たす ・負荷テストが実施されている ・パフォーマンステストが実施されている ・ネットワークダイアグラムが更新されている ・セキュリティアセスメントに合格している ・脅威分析試験に合格している ・障害発生時の復旧計画がテストされている ・(略)

図 3.1: 完成の定義の例

完成の定義を複数設けることもあります。たとえばスプリントでの完成の定義、リリースでの完成の定義のような形です。セキュリティ試験のように毎スプリント実施するのが難しいような項目を、リリースでの完成の定義にいれるわけです。一方で、このような形にすると、スプリントで完成しても、すぐには本番にリリースができないので、状況の変化に柔軟に対応できないこともあります。理想的には、スプリントでの完成の定義とリリースの完成の定義の差分は小さくなるようにしていきましょう。ただし、そうするためには、スクラムチーム全体の成熟が必要になります。

3.4 ドキュメントの扱い

アジャイルマニフェストでは、「包括的なドキュメントよりも動くソフトウェアを」としていますが、これはドキュメントが不要という意味ではありません。「左記のことがらに価値があることを認めながらも、私たちは右記のことがらにより価値をおく」とあるように、ドキュメント自体にも価値があります。

ドキュメントもプロダクトの性質によって求められるものが変わります。保守・運用の資料はひとたび本番としてリリースしたタイミングから使われることになりますが、最初になければいけないわけではなく、リリース前までに揃っていれば良いでしょう。一方でアーキテクチャーの全体像のようなドキュメントは共通理解を形成する上で、初期から継続的にメンテナンスしていく必要があるかもしれません。つまり、初期の段階で必要なものを明らかにしつつ、必要に応じて追加したり削除していくのがアジャイルなやり方です。また、ドキュメントの作成は必ず手作業でやらなければいけないわけではありません。自動化できるものは自動化しましょう。例えば、API ドキュメントのようなものは、ツールで自動生成できるので、継続的インテグレーションに組み込むことで、一切作業することなく最新のドキュメントを用意し続けることができます。

必ず作成・更新しなければいけないドキュメントがある場合は、完成の定義に含めたり、プロダクトバックログ項目を追加するようにしてください。

3.5　スプリントにおける活動

スクラムではコーディング作業とテスト作業は密接につながっています。つまり、コーディングやテストとやその他必要な作業が完成し、プロダクトオーナーの受け入れが終わって、はじめてその機能が完了したことになります。

完成の定義を満たしていない（非機能要求が未対応）もの、プロダクトオーナーが受け入れない（機能要求が実現されていない）ものは、スプリントでの成果と見なされません。

3.6　開発環境の作成

初期の段階では、開発に必要な環境の準備も必要です。まず、個人の開発環境を用意します。このとき、後から参加した人が簡単に開発環境を手に入れられるように仮想化や自動化などを組み合わせるようにしましょう。開発環境構築手順書を用意する方法もありますが、ドキュメントのメンテナンスコストや、実際の作業の手間を考えると無駄が多いと言えるでしょう。

個人の開発環境以外には、デモ環境やプロダクトオーナーの受け入れ確認環境も用意しておきましょう。そうすることで最初から継続的に完成したものをリリースできるようになります。そして、継続的インテグレーションサーバーやバージョン管理ツールなど、開発に必要なツールセットも用意しておきましょう。最初に用意しておくことで、統合された環境を作ることが可能になります。

ツールはメジャーなものを使います（図 3.2）。そうすることでインターネット上で多くの情報を入手できるようになります。

図 3.2: よく使われるツールの例

3.7　テスト自動化の準備

アジャイル開発では、ソフトウェアが動く状態を維持しながら機能を追加していきます。つまり、頻繁に何度もテストをしなければいけません。

小規模リリースのたびに手動でテストすると、コードベースが大きくなるにつれて、テストのコストがどんどん膨らんでいきます。こうなると開発できる機能の量が減っていき、それに伴ってビジネスの成果も減ってしまい良くありません。

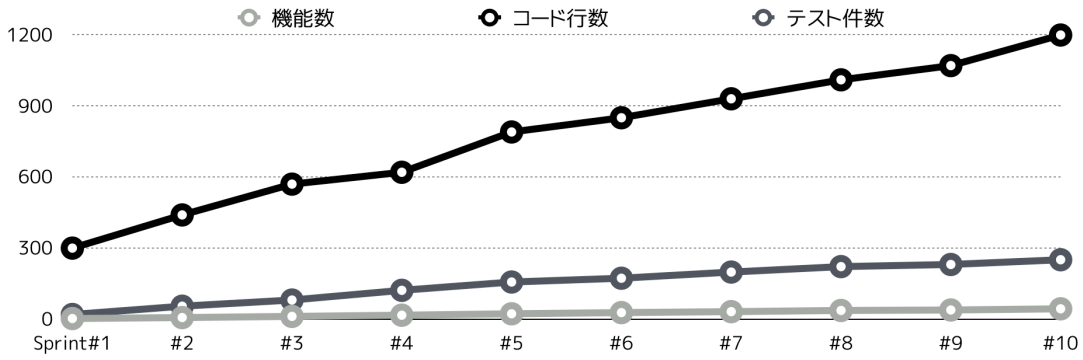


図 3.3: テスト自動化の必要性

スクラムではフレームワークを定義しているだけで、テスト自動化を含む技術的なプラクティスについては一切触れていませんが、事実上はテスト自動化が必須です。品質保証計画や非機能要件を踏まえて、どのようにテストを自動化していくか考えるようにしましょう。

3.8　プラクティスの選択

スクラムを適用する場合には、3 つのロール、3 つの作成物、5 つのイベントすべてを実施しなければいけません。

一方でエクストリームプログラミング（XP）では自分たちが必要なものを選べばよいことになっています。

初版(1999)【12】	2nd Edition(2004)【24】		現在?【19】	
	主要プラクティス (13)	**導出プラクティス (11)**	**共同のプラクティス (4)**	**管理者のプラクティス (5)**
計画ゲーム	全員同席	本物の顧客参加	イテレーション	責任の受け入れ
短期リリース	チーム全体	インクリメンタルなデプロイ	共通の用語	援護
メタファ(比喩)	情報満載のワークスペース	チームの継続	オープンなワークスペース	四半期ごとの見直し
シンプルな設計	いきいきとした仕事	チームの縮小	ふりかえり	ミラー
テスト	ペアプログラミング	根本原因分析		最適なペース
リファクタリング	ストーリー	コードの共有	**開発のプラクティス (6)**	
ペアプログラミング	週次サイクル	コードとテスト	テスト駆動開発	**顧客のプラクティス (4)**
共同所有	四半期サイクル	単一のコードベース	ペアプログラミング	ストーリー
継続した統合	ゆとり	デイリーデプロイ	リファクタリング	リリース計画
40時間労働	10分ビルド	交渉によるスコープ契約	ソースコードの共同所有	受け入れテスト
オンサイトのユーザ(顧客)	継続的インテグレーション	利用都度課金	継続的インテグレーション	短期リリース
コーディング規約	テストファーストプログラミング		YAGNI	
	インクリメンタルな設計			

図 3.4: XP のプラクティス

前述のとおりスクラムには技術プラクティスの定義はないので、XP などからほかに必要なプラクティスを選択するとよいでしょう。最初から盛り込みすぎると、学習コストが高すぎたり時間がかかりすぎたりするので注意します。また、一度プラクティスを適用したからといって変更できないわけではありません。機能しないプラクティスを外したり、もっとよいプラクティスが見つかれば置き換えてください。

第 4 章

計画やマネジメントの観点

次に、計画やマネジメントの観点を見ていきましょう。

4.1 見積りに影響するパラメータ

これから新しいプロダクトを開発する際に、スケジュールや費用をどうやって見積もればよいでしょうか。

それを考える上で、まず、これらの見積りに影響するパラメーターを見てみましょう。

- ビジネス上の都合（マーケティング、新製品発売、法令改正、売上確保など）
- 予算
- 開発チーム外への依存関係（QA、セキュリティ）
- 開発作業のボリューム（スコープ、完成の定義）
- 開発チームの能力（ベロシティ）

これらは相互に関連する要素で、同時にすべてを固定にすることはできません。したがって、ビジネス上の都合を優先するのか、開発のスコープを優先するのか、など何を優先すべきか共通理解を形成することが重要です。

また初期の時点では、いかなる数字の精度も低いので、スプリントを繰り返しながら適宜見通しを更新していくと良いでしょう。このとき、既に同じチームでほかのプロダクトの開発をしたことがある場合は、チームの能力や経験をそのまま生かせるので、新たなチームで始めるよりも見積り精度は高くなります。

4.1.1 チームのサイズが決まっている場合の費用の見積もり

チームのサイズが決まっている場合、1 スプリントあたりのチームにかかるコストはほぼ確定できます。例えば 1 ヶ月スプリントで PO と SM、開発チーム 6 人で、それぞれの人件費が 1 人 100 万円であれば、月間 800 万円の費用になり、1 スプリントあたりのコストはスプリントの期間に応じて算出可能で、以下の式になります (機材など諸経費は簡単のため省略)。

総費用 ＝ 1 スプリントあたりの費用 ×スプリント実行回数

(一方でスコープ網羅を目指して、スプリント回数が可変である場合は、プロジェクト初期では、何回スプリントをまわすべきかは確定できません。というのはチームが 1 スプリントあたりにどのくらいの量を作れるかの数値がないからで、その場合は、プロジェクト費用の見積もりはスプリントを経るごとに正確になっていくことになります)

4.1.2　期間を優先し、チームのサイズをあとから決める場合

アジャイルな開発手法の多くは、チームを継続的に維持することによって、チームの文化や規律や技術力を向上させることに価値を置いています。したがって寄せ集めチームを作ったりあとから人をどんどん追加していく方法はあまりうまくいきません。一方で既存のチームに対して文化が壊れない程度に人を追加する（もちろん Scrum のチーム人数の範囲で）というアプローチはないわけではありません。

この場合も実は前項の話とあまり変わりません。チームのサイズをいじったとしてもチームが 1 スプリントあたりに作れるモノの量は、プロジェクト開始時点では正確には分からないので見積り精度が向上するのは、スプリントを実行してからになります。

必要ベロシティ＝プロダクトバックログ項目の合計ポイント ÷ スプリント数

4.2　スプリントの長さを決める

実際にスプリントを開始する前には、スプリントの長さを決めておかなければいけません。スプリントは 4 週間までの固定の期間です。

期間を決定する上で影響を及ぼすのは以下のような項目です。

- 想定の期間
- 変化の度合い
- フィードバックの回数
- プロダクトの規模
- 開発チームのキャパシティや場所
- 完成の定義

たとえば、プロダクト開発の想定期間が 3 か月だとすると、4 週間スプリントではフィードバックサイクルが 3 周しか回らないことになるので、改善効果が出ません。また不確実性が高くなればなるほど、長い期間のスプリントでは途中の変化に脆弱になってしまいます。これらを踏まえて、小規模や短期間のものであれば 1〜2 週間とし、大規模であれば 2〜4 週間にすることが一般的です。なお、直感に反するかもしれませんが、スプリント期間が長いほうが難易度は上がります。

4.2.1 スプリントは短いほうがよい

1 チーム規模の開発であればスプリントの期間は 1 週間がお勧めです。その理由を列挙すると以下のようなものがあります。

- レトロスペクティブ（ふりかえり）が頻繁にあるので改善が進む
- 計画の精度が高くなる
- 例え失敗しても一週間で済むので実験しやすい
- ベロシティの数字がすぐ出るのでやる気になる
- 中だるみする余裕がない・リズムがよい
- 一週間で収まるサイズのプロダクトバックログ項目にするので明確な完成を定義しやすい。従ってプロダクトオーナーの受け入れを取りやすい
- スパイクが必要なプロダクトバックログ項目が明確になる
- プロダクトオーナーが変更を我慢しやすい
- ごまかしが効かない

順番に見ていきましょう。

レトロスペクティブ（ふりかえり）が頻繁にあるので改善が進む

大きな理由の 1 つがレトロスペクティブの回数です。 たとえば、スプリントに使える期間が 3 か月だとすると、4 週間スプリントはスプリントが 3 回、2 週間スプリントで 6 回、1 週間スプリントで 12 回となります。 改善の基本的な考えとして、うまくいったかどうかを判断しうまくいかない場合は元に戻す、一度にたくさんのことを変えないという点が挙げられます。 つまり毎回劇的な変化を加えるのではなく緩やかに変化していくことになります。 このとき改善の機会が少なければ少ないほど、試行錯誤して改善する機会が減ってしまい、最初と最後での変化は少なくなってしまいます （言い換えると上手くなる前に期限が来てしまいます)。 したがって、慣れていないうちほど、改善の機会をたくさん用意した方がよく、短いスプリントの方がその機会は多くなります。

計画の精度が高くなる

計画づくりは難しいです。もちろん見積りも難しいです。 そしてこれらは規模が大きければ大きいほど難易度が上がります。 また、始めたばかりの段階だと、スクラムチーム全体がやり方に慣れていない、過去にどれくらいの速度で開発できていたのかというデータがないといった要因も、影響を与えます。 したがって、特に慣れるまでは、なるべく小さい単位で計画をたてて、計画と実際の差がどのくらいなのかを見ながら微調整していくことが重要になります。 1 週間の計画を立てられないのに 4 週間の計画を立てられるということはないので、まずは短い範囲でいいので確度の高い計画を立てられるようにしてください。 そうすることで、プロダクトオーナーはプロダクトの今後の見込みや予測をしやすくなります。 予測がしやすければしやすいほど、ビジネス面での計画も立てやすくなります。

例え失敗しても一週間で済むので実験しやすい

計画の精度が高くなる、というのと多少相反しますが、万が一そのスプリントが何らかの理由でうまくいかなかったり、成果が少なかった場合でも、スプリントの期間が短ければ短いほど影響は少なくなります。 あるプロダクトバックログ項目でいろいろハマってしまい、想定の時間以上をかけても完成しなかった場合のことを考えてみましょう。 この場合、スプリント期間が長いと延々とハマってしまい、長いタイムボックスを使いきってしまうリスクもあります。 一方で、1 週間スプリントの場合は、すぐにスプリントが終了します。 そして着手中だろうがあとすこしで終わりそうだろうが強制的に当該プロダクトバックログの作業は停止されます。 仕掛中のプロダクトバックログ項目については、次のスプリントで継続して取り組むのか、それとも先送りにするのか、もうやめるのかを判断する機会もあります。 そう考えると短いスプリントの方がリスクが少ないことが分かります。

また、同様の理由で、プロダクトオーナー側の観点としても、挑戦的なプロダクトバックログ項目をスプリントに投入しやすくなります。

ベロシティの数字がすぐ出るのでやる気になる

これは大した理由ではないのですが、スプリントが短いとベロシティの数値がすぐに見える化されます。 過去のベロシティとすぐに比較できるので、成長を実感したり停滞に気づいたりするのが簡単になります。

中だるみする余裕がない・リズムがよい

1 週間スプリントでは、例えば以下のような週間スケジュールになります。

- 水曜日午後にスプリントプランニングを 2 時間のタイムボックスで実施し、完了後に夕方からスプリント開始
- 金曜日か月曜日くらいにプロダクトバックログリファインメントを実施して次のスプリントを準備
- 火曜日の夕方に翌日の準備をして、水曜日の午前中にスプリントレビューを 1 時間、スプリントレトロスペクティブを 1〜2 時間実施

毎週同じ曜日の同じ時間にイベントが行われることになるので、スケジュールを考えたり調整したりする必要がなく、リズムが出やすくなります。イベントをオープンスペースでできないような場合でも、単純に毎週繰り返しで会議室予約をしておけばよいだけです。何曜日の何時くらいにどういう状態だったら、マズイとか大丈夫そうだ、というのも分かりやすくなります。スプリント期間が長くなると、スプリントプランニングのあとや翌日などはペースダウンしがちです。またスプリントの後半に向けて追い上げ型にもなりがちです。ところが、1 週間の場合は後半も何も、そもそも時間が短いので追い上げが効かず、いつも同じペースで着実にこなしていくことになります。

一週間で収まるサイズのプロダクトバックログ項目にするので明確な完成を定義しやすい。従ってプロダクトオーナーの受け入れを取りやすい

スクラムでは、スプリント単位で、完成の定義と受け入れ基準にしたがってプロダクトバックログ項目を完成させなければいけません。プロダクトバックログ項目が大きいので、複数スプリントにまたがって 1 つのプロダクトバックログ項目に着手しようとしてはいけないのです。つまり、スプリント期間が短ければ短いほど、プロダクトバックログ項目のサイズは小さくなるのが普通です（1 つのスプリントに投入するプロダクトバックログ項目が 1 つということはなく、大きくてもスプリント期間の半分以下で終わるようなサイズの項目を複数個投入するのが一般的です）。時間が短くプロダクトバックログ項目が小さいということは必然的に中身が明確であることにつながります。

スプリントに投入したプロダクトバックログに関してスプリント開始後にたくさん不明点が出てくるような状況だと、そのプロダクトバックログ項目は完成しない確率が高くなります。長いスプリント期間であれば、とはいえ、なんとか頑張って吸収するということができるかもしれませんが、それは本来やるべき事前準備不足を隠してしまうことにもなります。一方で、1 週間スプリントだと、もし不明点がでてその時すぐにプロダクトオーナーと会話ができないような待ちが発生するとどうにもならなくなります。したがって、プロダクトバックログリファインメントで事前に次以降のスプリントに投入される可能性のあるプロダクトバックログ項目が本当に Ready なのかを精査するようにもなります。そして 1 週間の半分くらいの規模のプロダクトバックログ項目であれば、「なにができれば完成なのか」の認識は比較的容易に合わせられるはずです。認識があっていれば、プロダクトオーナーの受け入れ確認も容易になります。

スパイクが必要なプロダクトバックログ項目が明確になる

上の項目に関連して、スプリント期間が長いほど、あるプロダクトバックログ項目を実装するのにあたって、調査系のタスクも一緒に含めてしまいがちです。完成とは何なのかは合意できていても、それをどうやるのかはスプリントに入ってから調査して進めていってしまうのです。一方で、1 週間スプリントのような短い期間では、特定のプロダクトバックログに関する調査をスプリントに入ってからやっていたのでは、プロダクトバックログ項目が完成しないリスクが高くなります。つまり事前に調査まで終わらせておいて、開発チームが自信をもって作れる状態までもっていくような力が働きます。プロダクトバックログリファインメントなどで、調査が必要なプロダクトバックログ項目を明らかにして、バッファを使って事前調査したり、場合によってはプロトタイプなどを作るプロダクトバックログ（タイムボックス上限を定める）を事前にこなすようになっていきます。こうすることで、プロダクトバックログ項目がスプリント期間中に完成する確率が上がっていきます。

プロダクトオーナーが変更を我慢しやすい

プロダクトオーナーはステークホルダーと協働し、開発チームのパフォーマンスを踏まえて、プロダクトをどうしていくかシビアな判断が必要になります。そして、ビジネスの状況は刻一刻と変化します。 長いスプリントになればなるほど、ステークホルダーからの要求やビジネスの状況との乖離の影響を受けて、プロダクトバックログ項目を入れ替えたいという欲求が強まります。一方で、スプリント

期間中のプロダクトバックログ項目の入れ替えは、プロダクトオーナーと開発チーム間の調整が必要です。単に追加のプロダクトバックログ項目を依頼したり、着手中のプロダクトバックログ項目の受け入れ条件を変更したり、といったことは開発チームの合意なしではできません。一方で、プロダクトオーナーとしては変更したい理由があるのに、それが受け入れられないのを我慢しなければいけないことにもなります。

ここでスプリント期間が短ければ、現状は受け入れた上で、次のスプリントで軌道修正をかければよいと割り切ることができます。その割り切りによって開発チームは割り込みを受けること無く集中して作業を進められます。

なお、プロダクトオーナーにはスプリントをキャンセルする権限が認められているので、どうしてもという場合はスプリント期間に関わらずスプリントを途中中止して構いません。ただし、いつもそれをやっているとプロダクトは何もできず、開発チームのモチベーションにも影響があるので、数か月に 1 度といった頻度までが限界です。プロダクトオーナーとしての準備を怠ったが故のキャンセルは許されません。

ごまかしが効かない

ここまで見てきた内容全体によって、スプリント自体のごまかしが効きにくくなります。 事前準備をきちんと行い、ムダなく着実に進めていかないと、成果がでない状況になります。 つまり、長いスプリントではなんとか吸収しきれていた問題は、短いスプリントだと大きな問題として露見していくことになります。 長いスプリントによって隠れていた本当の問題が露見すれば、それを改善することで、より成果が出やすくなります。

4.2.2　1 週間スプリントの弊害

1 週間スプリントの利点ばかり説明してきましたが、いくつかの弊害や考慮ポイントがあります。 最後にそれを説明しておきます。

複数開発チームがある状況では、結合が間に合わない

それなりの規模の開発で、開発チームが複数ある場合、1 週間のスプリントではそれぞれの開発チームの成果物の結合が間に合わない可能性があります（Nexus などでも毎スプリントの結合が求められています）。 複数開発チームが存在する場合、コミュニケーションのオーバーヘッドがどうしても避けられず、またどこか一箇所の問題が全体に影響を与えやすくなります。 したがって、バッチサイズを多少大きくして確実に結合していく必要がでてくるかもしれません。

いつも追い立てられている気がする・ゆっくり考える時間がない気がする

1 週間という短い期間に区切ると、どうしても残り日数や時間を意識する回数が増えます。 したがって人によっては、いつも追い立てられていると感じることもあります。 持続可能なペースで働くことが重要なので、その場合は、バッファの比率を見直すのも 1 つの手です。 また、プロダクトの状況次第では、祝祭日の絡む週のスプリントをときどきスキップする、という選択もできるかもしれません（あ

まりお勧めはしませんが)。 別の要因として、外部からプレッシャーがかかっていることもあります。持続不能なペースで働くこと、決めたことを全て達成することを約束しているわけではないので、組織内で別のよくない力学が働いていないかどうかを確認し、それを改善していくとよいでしょう。

4.3 リスクを管理する

プロダクトの開発を始めた初期の段階では不確実なこと、わからないことだらけです。それらを見ないふりをして先に進めると、途中で大きな問題になってしまうことがあるので、なんらかの形でリスクをバックログ化、見える化しておくと良いでしょう。このバックログは、プロダクトオーナーとスクラムマスターが中心となって対処していきます。

#	リスクの内容	顕在化確率	発生時の影響日数	ポイント
1	接続先システムとのインターフェースがあわない	50%	10	5
2	マスタデータの提供が遅れる	30%	30	9
3	社内品質管理部門の指摘	40%	10	4
4	専用線の調達の遅延	60%	20	12
5	ブラウザ間の非互換性	30%	5	1.5
6	App Storeからのリジェクト	40%	10	4

図 4.1: リスク管理の方法の例

4.4 報告

組織の活動の一環としてプロダクトを開発する以上、なんらかの報告やレポートが必要になることが普通ですが、組織によって必要な内容は異なります。初期の段階で、どんな頻度で、どんな内容を、誰にレポートしないといけないのかを明らかにしておきましょう。

報告の内容は、全体像やリスクを把握できるようなものにとどめる（最初は最低限から始める）ようにしましょう。プロダクトバーンダウンチャートのようなものでも十分なことも多々あります。スプリントの活動を詳細にレポートするのは意味はありません。あくまでもプロダクトの観点やプロダクトに関係するリスクの観点に留めるようにします。

開発チームの時間を報告作業に使うと勿体無いので、自動で作るか、スクラムマスターやプロダクトオーナーが作成するのも良いでしょう。また、レポートに対するフィードバックやそれを受けてのアクションがないなら、レポート自体を廃止するようにして、無駄をなくすようにします。

4.4.1 メトリクスの間違った活用に気を付ける

報告のときにあわせて関心を持つのがメトリクスです。前述のとおり、外部への報告はあくまで全体像やリスクを把握できるようなものにとどめるべきですが、スクラムチーム自身の改善のためにはメト

リクスは有効です。逆に、スクラムチーム外から監視や人事評価やプロジェクト評価のために、チームのためのデータを使うと、以下のようなことが発生するので避けるようにしましょう。

ベロシティの数値そのものの大小が評価される

スプリントプランニングで、見積りが上振れする（インフレを起こす）ことにつながります。あくまでベロシティは、このペースでいくといつ終わるのか、もしくはスプリント数が限られている場合は上からどのくらいまでのプロダクトバックログ項目ができそうか、という予測のツールでしかない点に注意してください。

スプリントのタスクにおいて、残り時間だけでなく、見積もり時間と実時間の乖離が測定され評価される

バッファを積んだ安全な見積もりを行い、早くタスクが完了しても、予実の差を発生させないように、時間が来るまで完了にさせないようになります。知りたいのはそのスプリントプランニングで選択したプロダクトバックログ項目を完成できるかどうかであって、かかった時間を追跡するのは無意味です。今後同じ作業をする場合に参考になるから、という話を聞くこともよくありますが、スプリントの回数をこなせば見積もりの精度は徐々にあがるとともに、レトロスペクティブ等で改善活動も行えます。かかった時間を測定する時間はムダではないのか？ をよく考えてください。もしかしたら使うかも！ の作業は本当にやるべきでしょうか？

スプリントのタスクをこなした数で評価を行う

開発チームのメンバーが自分ができる簡単なタスクだけを取りたがり、他のメンバーを助けないようになります（助けてもインセンティブがないと感じてしまうからです。本当は開発チーム全体の責任にも関わらず…）。

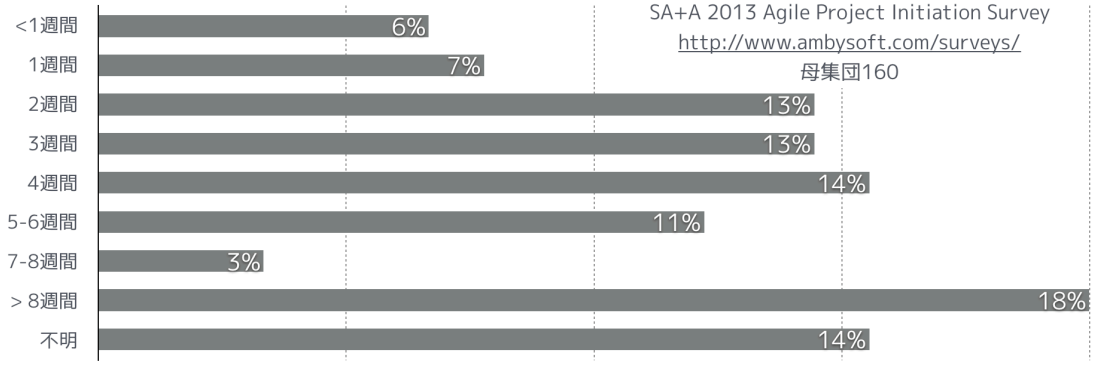
バグの発生件数の多寡で評価される

バグを隠すようになります。プロダクトの評価は、そのプロダクトをマーケットに投入することで、どれくらいビジネス価値が生まれたのか、それは投資したコストに見合ったものなのかで評価されるべきであって、その評価抜きに開発チームを評価しても仕方ありません。

第 5 章

初期フェーズの進め方

多くのプロダクト開発では事前に 1〜2 か月の準備をしています。



ただし、ここまで説明した全ての項目の検討を行わないといけないわけではありません。最初にタイムボックスを決めて、その範囲の中で行うようにしましょう。

つまり、検討すべき項目や実施すべき項目を洗い出して並び替えて、それぞれの項目は何をもって完了とするかを決めてから着手するようにします。並び替えた上位の項目から進めていき、進める過程で検討項目を追加したり、削除しましょう。スプリント開始後でよいものは初期にやる必要はありません。

つまり、準備期間もスクラムの考え方で進めていくことが基本です。厳密である必要はありませんが、スプリントプランニング、デイリースクラム、レトロスペクティブなどの要素を準備に取り込んで行きましょう。

付録 A

スクラム関連用語集

A.1　基本用語

プロダクトオーナー

開発するプロダクトにおける責任者である。そのプロダクトが実現するビジネス価値に対して責任を負う。 主な役割としてプロダクトのビジョンを明らかにして周りに伝える、プロダクトバックログのメンテナンス、プロダクトバックログ項目の優先順位付け、リリース計画の立案、開発チームが作成したインクリメントを受け入れるか受け入れないかの判断、ステークホルダーとの調整などをおこなう。実施中のスプリントをキャンセルすることができる唯一の存在である。 プロダクトオーナーはスクラムマスターと兼任しないことが強く推奨される。

開発チーム

実際にプロダクトを開発するメンバーのことを指す。 開発チームは、例えばデータベースの管理、アプリケーションの開発、テストなどプロダクトを作るために必要なことは全てをおこなえる必要があり、このことをクロスファンクショナルと呼ぶ。 開発チームでの仕事のやり方は開発チームのメンバー同士で合意して決めることができる。 スクラムにおいては開発チームの人数は 3 人〜9 人までとされており、開発チームのメンバーは同じ場所に居なければならない。 これ以上の人数で構成される場合は開発チームの分割をおこなう。

スクラムマスター

スクラムチームがスクラムの価値を理解しプロセスを正しく実践できることに責任を負う。 主な役割としてコーチング、ファシリテーション、チームへの奉仕的な活動などがある。 またプロダクトオーナーと開発チームが円滑なコミュニケーションできるようにしたり、外部からの妨害（例えば割り込みタスクやスプリントで実施するプロダクトバックログ項目の変更や追加など）からチームを守ったりする責任がある。 スクラムマスターはプロダクトオーナーと兼任しないことが強く推奨される。

スクラムチーム

プロダクトオーナーとスクラムマスターと開発チームの集合のこと。

ステークホルダー

プロダクトの利用者やスポンサー、社内の上司や他部門などの、プロダクトに対して利害関係を持つスクラムチーム以外の人を指す。 スクラムにおいてはプロダクトの機能や構築の優先順位などはプロダクトオーナーが最終決定権限を持ち、ステークホルダーは直接開発チームに機能の構築や要件の変更を指示することはできない。 あくまでプロダクトオーナーに対して伝えることになる。 スプリントレビューに参加してプロダクトに対してフィードバックすることは可能であり、良いプロダクトを作る上で、そのフィードバックはとても重要である。

スプリント

4 週間以下のタイムボックス。この期間内に開発チームは要求事項を「完成」させ、動作するリリース判断可能なインクリメントを作る。 スプリントは 4 週間以内の短いプロジェクトとも考えることができる。 スプリントが終了したら、進捗や状況などを正確に把握し、状況に応じた調整をおこなった後、速やかに新しいスプリントを開始する。

スプリントプランニング

スプリント内での作業を計画するイベント。 スプリントプランニングにはトピックが 2 つある。 1 つ目のトピックでは、スクラムチームはプロダクトバックログの上位からスプリントでどの項目を実現するかを理解する。 プロダクトバックログからどれくらいの項目を実現するかは、開発チームのこれまでの作業実績（ベロシティ）から予測を立て、最終的には開発チームが責任を持つ。 2 つ目のトピックでは、開発チームが中心になって選択された項目をどう実現するかを作業レベルまで分解して、より具体的で詳細な計画を作成する。 なお、トピック 2 で詳細に計画を立てた結果トピック 1 で取り上げたプロダクトバックログ項目はやはり実現できないといった判断をすることもある。

デイリースクラム

スプリントが問題なく進んでいるか検査し、何かあれば即座に解決を促すためのミーティング。 毎日、同じ場所にて、同じ時間から開始し 15 分以内で終了する。立ったままスクラムボードの前でやることも多い。 このまま進めて当初の計画どおりに完成するのかを確認する。 スプリントのゴールを達成するために昨日やったこと、今日やること、困っていることを開発チーム全員が話すこともあるが、このフォーマットは必須ではない。解決に時間がかかるような問題の解決や議論の場ではないので、そのような内容は別の会議を設ける。

スプリントレビュー

スプリントの最後に実施されるイベント。 スプリントの成果物をステークホルダーに示しフィードバックを得る。 必要があれば今後のスプリントの進め方について話しあい、プロダクトバックログを調整する。 スクラムチームはスプリントで完成した成果物だけをデモできる。最大の目的は参加したステークホルダーからフィードバックを得て、それによってプロダクトの価値をさらに向上すべく、今後のプロダクトバックログを見直すことにある

スプリントレトロスペクティブ

スプリントの最後にスクラムチームの改善のためにおこなうイベント。 改善のためにおこなうので実行可能な改善点を出すことが求められ、次のスプリントで実行されたかどうかを評価する。 どのようなやり方をするのかはスクラムでは決められていない。 KPT というやり方がよく使われるが、やり方はこれ以外にも多数あり、状況に応じて選択すると良い。 多数の改善案を出しても一度に変更することは評価を難しくしてしまうため、大きな効果がありそうなことに絞ると良い。

プロダクトバックログリファインメント

プロダクトバックログ項目は、スプリントで着手する前までに準備ができた状態になっていなければいけない。 そのため、前のスプリントなどで事前に時間をとって、上位のプロダクトバックログ項目の中身を見て、疑問点がないかを確認し、見積りを済ませておく。 この活動はいつ実施するか決められていないが、スプリントの中間くらいの時期にやることが多い。

プロダクトバックログ

プロダクトを作成するにあたっての要求事項を順番に並べたリスト。 スクラムではリストの上から順番に終わらせていく。 一般的にはビジネス価値の高いものが上位に来る傾向にある。 直近のスプリントで手を付けるプロダクトバックログ項目は詳細に記述されており、リストの末尾にいくに従ってあまり詳細ではなくなる。定期的に見直す必要がある。 順位の並べ替えについてはプロダクトオーナーが責任を持ち、他のロールによって順番が変えられることはない（変更の提案は可能）。

スプリントバックログ

スプリントで実施するプロダクトバックログ項目のサブセットと、それを分析し具体的なタスクに落とし込んだもの。 プロダクトバックログでは通常、実装に関する記述をおこなわないが、スプリントバックログは開発チームがそのスプリントを完成させるために必要な作業であるため具体的に記述する。 個々のスプリントバックログ項目は、一般的に最大 8 時間程度のサイズより小さなサイズに分割することが望ましい。 また、理想時間（チームの平均的な能力のメンバーが集中してそれに取り組んだ場合にかかるであろう時間）によって見積もることも多い。

インクリメント

完成の定義を満たしており、リリースしようと思えばリリースできるプロダクトのこと（実際にリリースするかどうかはビジネスの状況などによる）。特定の部品や技術層だけを作り込んでも、利用者に価値がなければ意味がなく、一気通貫で利用者に役立つことが求められる。

A.2 関連用語

完成の定義

プロダクトとして定めた「リリース判断可能なプロダクト」を作成するために実施しなければいけないことの一覧。例えば、ユニットテストがある、カバレッジ 80% 以上である、統合テストがおこなわれている、リリースノートが書かれている、レビューがおこなわれているなどである。プロダクトバックログ項目の完成の定義、スプリントの完成の定義、リリースの完成の定義など、複数に分割することもある。完成の定義は作っているプロダクトによって異なる。開発を進めていく中で完了の定義を縮小することは許容されない。

スプリントバーンダウンチャート

スプリントの進捗を可視化し課題発見のきっかけにするためのチャート。横軸にスプリント内の日数、縦軸にスプリントバックログ項目の残り作業の見積り時間の合計を毎日プロットする。毎日決まった時間（デイリースクラム前にやることが多い）に更新する。チャート上には理想線と呼ばれる理想の推移線を引き、この理想線に対して実際の推移がどのようになっているかを比較していくことで開発チームの進捗が明らかになる。理想線と比較してバーンダウンチャートが収束に向かわない場合はスクラムマスターを交えて開発チームで話しあい早めに解決方法を考える。

スクラムボード

スクラムでプロダクトを開発する際にチームの道具として用意するボードのこと。プロダクトバックログ項目やスプリントバックログ項目、バーンダウンチャート、妨害リストなどを掲載する。全員同席しているスクラムチームの場合は壁やホワイトボードを使って作成する。それによって見える化を促進する。特に決められた形式はないのでスクラムチームは自分たちの必要性に応じて掲載するものを増やしたり、レイアウトを変更したりして良い。

相対見積り

プロダクトバックログ項目の見積もる際に、具体的な人月や人日で見積もるのではなく、小さめのプロダクトバックログ項目と比較してその何倍なのかという観点で見積もる方法。プランニングポーカーと呼ばれるカードを使って開発チーム全員で見積もる。見積りの結果についてはプロダクトオーナー

が干渉することは許されない。 時間などの物理単位で見積もった場合には、開発チームのメンバーによって見積りが大きく異なることになったり、見積りの誤りが計画に与える影響が大きくなったりするが、相対的に見積もることでそれらの問題を解決する。

受け入れ基準 （Acceptance Criteria）

プロダクトオーナーの視点から何を持ってプロダクトバックログの項目が完成したかを確認するための基準。インクリメントは「完成の定義」と「受け入れ基準」の両方を満たして、はじめて完成となる。ユーザーストーリー形式で要求事項が書かれている場合は、実際のユーザーの操作をシナリオ形式に記述したデモ手順を受け入れ基準にすることもある。

スプリント 0

実際のスプリントを開始する前におこなう、開発をスタート可能にするための準備期間のこと。プロダクトバックログの準備、開発環境の準備、スクラムチームの作業場所の準備、完成の定義の作成、技術的な検証などをおこなう。このフェーズではリリース判断可能なプロダクトが作られないため長い時間をとることは推奨されない。おおよそ第 1 スプリントの開発をおこなうのに十分になれば良い。

ベロシティ

開発チームの速度を表す指標で、スプリントで「完成」したプロダクトバックログ項目の見積りポイントの合計値のことである。スプリントの実施回数が決まっている場合は開発チームが作成できる機能の合計サイズ＝ベロシティ×スプリント数となり、逆に全ての機能を作成する場合は、スケジュール＝総見積りポイント数÷ベロシティとなる。直近 2〜3 スプリントの平均値を開発チームの速度として使うケースが多く、改善を進めていくと開発チームのベロシティは向上していく。

ユーザーストーリー

顧客がソフトウェアで実現したいことを詳細に文章化するのではなく、ソフトウェアを利用する実際のユーザーに何をさせたいかに焦点を当てて簡潔に要求事項を表す書き方である。スクラムで規定された書き方ではないが、採用例が多い。 各要求事項で達成したいゴールとプロダクトにもたらす価値の記述に注力することで、適切なタイミングでの開発チームとプロダクトオーナーの会話を促す。これにより、文書化に頼りすぎない進め方がし易くなる。

妨害事項リスト（Impediments List）

各ロールの責務が果たせていない場合や、正しくスクラムイベントが実施されていないなどのスクラムで規定されていることができていない事象や改善すべき事項を妨害事項と呼ぶ。これらを管理した一覧表を妨害事項リストと呼び、日々見つかった妨害事項が記載されている。スクラムマスターがこの一覧を管理し、記載された項目に順序を付ける。そして、その順番に基づいて妨害事項を解決していく。

なお、これは必須の作成物ではない。

Scrum of Scrum

複数の開発チームでスクラムを実施する場合に、プロダクト全体で解決すべきことがあるかを確認するために実施される。各開発チームのデイリースクラム後に開発チームの代表とスクラムマスターが集まり、単一の開発チームだけでは解決できない課題の有無を確認し、何か問題があればすぐに解決のためのアクションや議論する会議を設定する。また、組織的にスクラムを導入している現場では、各プロダクトで解決できない課題を確認する会議として実施されることもある。

XP（eXtreme Programming）

正式名称はエクストリームプラグラミング。ケント・ベックらによって提唱されているアジャイル開発手法の 1 つ。開発者が日々おこなうべきプラクティス（実践すべき項目）が多く規定されているのが特徴である。スクラムはプロセスにフォーカスを当てているため、開発者がすべきこととして XP のプラクティスを組み合わせて実施されることが多い。代表的なプラクティスとしては、ペアプログラミング、テスト駆動開発、継続的インテグレーションなどである。

Scrum Starter Guide

2019 年 12 月 1 日　初版第 1 刷　発行

著　者　吉羽龍太郎 (@ryuzee)