アジャイル開発 解説書

ソフトウェア開発を迅速かつ柔軟に進めるための手法である「アジャイル開発」の基礎から実践、改善手法までをわかりやすく解説する手引きです。

章構成

- **1. 入門編：アジャイル開発の基礎と考え方**

  アジャイル開発が生まれた背景、従来のウォーターフォール開発との違い、アジャイル宣言と12の原則について解説します。

- **2. 実践編：スクラムフレームワークとチーム開発**

  最も普及しているアジャイル手法であるスクラムの役割、イベント、作成物と、日々の開発の進め方を解説します。

- **3. 実践編：XPとエンジニアリングプラクティス**

  高品質なソフトウェアを維持し続けるための技術的習慣（TDD、ペアプログラミング、CI/CD、リファクタリング）を解説します。

- **4. 応用編：ふりかえりと継続的改善**

  チームが自律的に成長するためのふりかえり（KPT等）の手法、メトリクスの活用法、よくある失敗パターンと対策を解説します。

入門編：アジャイル開発の基礎と考え方

アジャイル開発の基本的な概念、従来の開発手法との違い、土台となる価値観と原則を解説します。

アジャイル開発とは

アジャイル開発とは、ソフトウェアを小さな単位に分けて素早く開発し、変化に柔軟に対応しながら進める手法の総称です。「アジャイル（Agile）」には「俊敏な」「機敏な」という意味があります。

従来の「ウォーターフォール開発」では、計画、設計、実装、テストを段階ごとに順番に進めます。この方法では、途中で仕様の変更が発生した場合に対応が難しく、開発期間の終盤まで動くソフトウェアを確認できません。

アジャイル開発では、1週間から数週間の短い期間（イテレーションまたはスプリント）ごとに、計画からテストまでを一通り行い、実際に動くソフトウェアを完成させます。短い期間で成果物を確認しながら進めるため、ユーザーの意見を取り入れやすく、計画の修正も容易です。

アジャイルソフトウェア開発宣言

2001年に米国の技術者たちがまとめた「アジャイルソフトウェア開発宣言」では、4つの重要な価値基準が提示されています。

- **プロセスやツールよりも個人と対話を**

  決まった手順や道具を重視するよりも、チームメンバーや関係者同士の直接的な会話と協調を大切にします。

- **包括的なドキュメントよりも動くソフトウェアを**

  膨大な仕様書を作成することに時間をかけるよりも、実際に動作して価値を提供するソフトウェアを作ることを優先します。

- **契約交渉よりも顧客との協働を**

  契約条件に縛られて対立するのではなく、利用者の課題を解決するために顧客と協力して開発を進めます。

- **計画に従うことよりも変化への対応を**

  最初に立てた計画を頑なに守るよりも、状況の変化や新しい要求に合わせて柔軟に計画を見直します。

右側の事柄に価値がないわけではありませんが、アジャイル開発では左側の事柄により大きな価値を置きます。

アジャイルの12の原則

アジャイル宣言を具体的に実践するための行動指針として、12の原則が定められています。

- 顧客満足を最優先し、価値あるソフトウェアを早く継続的に提供する
- 開発の終盤であっても要求の変更を歓迎する
- 動くソフトウェアを、数週間から数か月の短い間隔で頻繁に届ける
- ビジネス側の人々と開発者は、プロジェクトを通じて日々一緒に働く
- 意欲ある人々を集めてプロジェクトを構成し、環境と支援を与えて信頼する
- 情報を伝える最も効率的で効果的な方法は、対面での会話である
- 進捗の最も重要な尺度は、動くソフトウェアである
- 持続可能な開発を維持し、一定のペースを保ち続ける
- 優れた技術と優れた設計に継続的な注意を払うことで俊敏性を高める
- シンプルさを重視し、作らない作業の量を最大化する
- 最良のアーキテクチャや要求、設計は、自己組織的なチームから生まれる
- チームが定期的に効率を高める方法をふりかえり、やり方を調整する

ウォーターフォール開発との比較

開発手法の選択を正しく行うために、2つの手法の特徴を整理します。

項目	ウォーターフォール開発	アジャイル開発
開発単位	プロジェクト全体を一括で進める	小さな機能単位で反復して進める
計画と変更	初期の計画を厳格に維持する	変更を前提に計画を柔軟に見直す
成果物の確認	開発終盤のテスト工程で確認	各反復の終了ごとに動くものを確認
主な適用領域	要件が明確で変更が少ないシステム	要求が変化しやすいサービスや新規事業

アジャイル開発は、ユーザーの要望が途中で変わる可能性が高い現代のソフトウェア開発において、リスクを最小限に抑えるための有効な選択肢です。

実践編：スクラムフレームワークとチーム開発

アジャイル開発で最も広く使われているフレームワークである「スクラム」の役割、イベント、作成物について解説します。

スクラムの基本概念

スクラムは、複雑な課題に対してチーム全員で協力して価値を生み出すための軽量な枠組みです。スクラムは「経験主義（透明性・検査・適応）」に基づいています。

- **透明性**

  開発の状況や課題を全員が見える状態にします。

- **検査**

  成果物や進捗状況を定期的に確認し、問題がないか調べます。

- **適応**

  検査で見つかった問題に対して、素早く手順や計画を修正します。

スクラムの3つの役割

スクラムチームは通常10人以下の少人数で構成され、階層構造を持たない自律的なチームです。チーム内には3つの役割が存在します。

- **プロダクトオーナー (PO)**

  プロダクトの価値を最大化する責任を持ちます。何を作るべきかの優先順位を決め、要求一覧（プロダクトバックログ）を管理します。

- **スクラムマスター (SM)**

  スクラムの理論と実践が正しく行われるようチームを導く責任を持ちます。チームの障害物を取り除き、円滑な活動を支援します。

- **開発者 (Developers)**

  各スプリントで使用可能な成果物（インクリメント）を作成する実務メンバーです。設計、プログラミング、テストなどの専門スキルを持ち寄ります。

スクラムの5つのイベント

スクラムでは、規則正しいリズムを作るために一定期間の固定された会議（イベント）を実施します。

- **スプリント**

  1週間から最長1か月間の固定された開発期間です。スプリントの期間中は他の4つのイベントがすべて含まれます。

- **スプリントプランニング（計画）**

  スプリントの開始時に行います。今期のスプリントで何を作成し、どのように作業を進めるかを決定します。

- **デイリースクラム（朝会）**

  毎日同じ時間と場所で15分程度行います。前日の成果、当日の予定、進行を妨げている障害を共有し、目標に向けた調整を行います。

- **スプリントレビュー（成果確認）**

  スプリント終了間際に行います。完成した動くソフトウェアを関係者に見せ、フィードバックを受け取ります。

- **スプリントレトロスペクティブ（ふりかえり）**

  スプリントの最後に行います。チームの作業手順や関係性を振り返り、次のスプリントで実施する改善点を決めます。

スクラムの3つの作成物

作業と成果物の透明性を担保するために、スクラムでは3つの作成物を定義しています。

- **プロダクトバックログ**

  製品に必要なすべての機能、改善要望、修正項目を優先順位順に並べた一覧です。

- **スプリントバックログ**

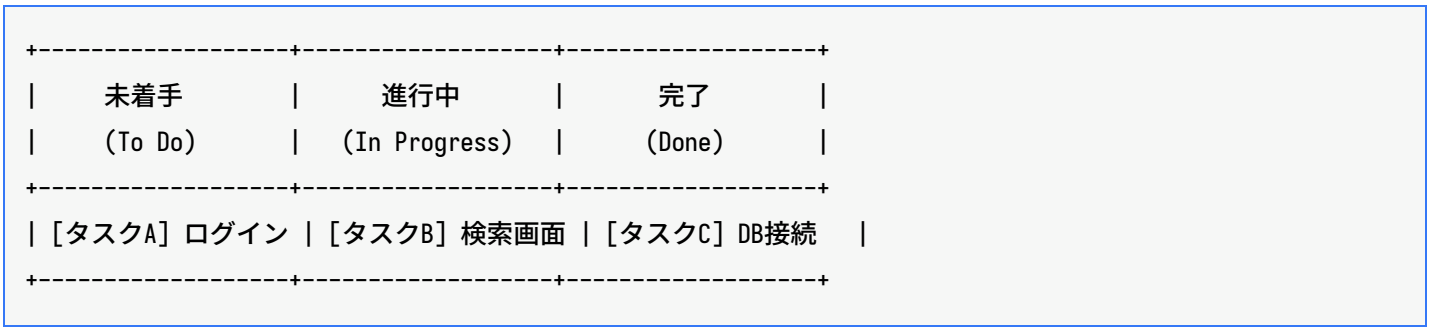
  現在のスプリントで達成する目標（スプリントゴール）と、それを実現するために選ばれた作業項目の集まりです。

- **インクリメント**

  スプリント期間中に完成した、実際に動作して利用可能な製品の増分です。「完成の定義（DoD）」を満たしている必要があります。

タスクの可視化と進捗管理

チーム全員が状況を一目で把握できるように、カンバンボードやバーンダウンチャートを活用します。



タスクの状態を移動させることで、誰がどの作業を担当し、何がボトルネックになっているかをチーム全員で素早く共有できます。

実践編：XPとエンジニアリングプラクティス

アジャイル開発において、高い品質と開発速度を維持し続けるための具体的な技術的実践（エンジニアリングプラクティス）を解説します。

エクストリーム・プログラミング (XP) とは

エクストリーム・プログラミング（XP）は、ソフトウェア開発における優れた習慣を極限（エクストリーム）まで高めて実践するアジャイル手法です。スクラムがプロジェクト管理やチーム体制に焦点を当てているのに対し、XPはコードの品質や開発手法そのものに重点を置きます。

テスト駆動開発 (TDD)

テスト駆動開発（TDD: Test-Driven Development）は、プログラムを書く前にまず自動テストを作成し、そのテストを通過させるように最小限の実装を行う手法です。

TDDは以下の3つのステップ（レッド・グリーン・リファクタリング）を素早く繰り返します。

```
[1. レッド] 失敗するテストを書く
      ↓
[2. グリーン] テストに合格する最小限のコードを書く
      ↓
[3. リファクタリング] 動作を維持したままコードを整理する
      ↓
   （繰り返し）
```

- 1. レッド (Red)

  これから作りたい機能の仕様を表すテストコードを書き、実行して失敗させます。

- 2. グリーン (Green)

  テストを通過させるための最短・最小限の実装コードを書きます。

- 3. リファクタリング (Refactor)

  テストが通る状態を保ちながら、コードの重複を取り除き、読みやすく保守しやすい構造に整えます。

TDDにより、常に自動テストで安全性が保障されるため、恐れずにコードを変更できるようになります。

ペアプログラミングとモブプログラミング

複数人で協力してコードを書くことで、品質向上とチーム内の知識共有を同時に実現します。

- **ペアプログラミング**

  2人の開発者が1台のPCを使って開発します。1人がコードを入力する「ドライバー」を務め、もう1人が全体の設計や誤りをチェックする「ナビゲーター」を務めます。役割は定期的に交代します。

- **モブプログラミング**

  チーム全体（3人以上）で1つの画面を見ながら開発を進めます。全員の知恵が集まるため、属人化を防ぎ、仕様の合意形成を迅速に行うことができます。

継続的インテグレーションと継続的デリバリー (CI/CD)

- **継続的インテグレーション (CI)**

  開発者が書いたコードを1日に何度もメインブランチに統合し、自動テストとビルドを実行して問題を即座に検知する仕組みです。

- **継続的デリバリー / デプロイ (CD)**

  CIを通過したコードを、ステージング環境や本番環境へ安全かつ自動的にリリースできる状態を維持する仕組みです。

ユーザーストーリーと受け入れ基準

要求を開発者が理解しやすい形式で定義するために、「ユーザーストーリー」と「受け入れ基準」を用います。

ユーザーストーリーは以下の形式で記述します。

> **［ユーザーの種類］として、**
> **［達成したいこと］をしたい。**
> **なぜなら［得られる価値］だからだ。**

受け入れ基準には、Given-When-Then 形式を使うと動作条件が明確になります。

> **前提（Given）：ユーザーがログイン画面を開いている**
> **もし（When）：正しいメールアドレスとパスワードを入力して送信ボタンを押す**
> **ならば（Then）：マイページに遷移し、「ようこそ」と表示される**

応用編：ふりかえりと継続的改善

アジャイル開発においてチームを自律的に成長させるための「ふりかえり」の手法、計測指標、よくあるアンチパターンとその対策を解説します。

ふりかえり（レトロスペクティブ）の重要性

アジャイル開発では、定期的に自分たちの働き方を点検し、改善し続ける「カイゼン」の姿勢が不可欠です。スプリントの終了時に実施するふりかえりでは、誰かを責めるのではなく、「仕組みやプロセスをどう改善できるか」に焦点を当てます。

KPT法によるふりかえりの進め方

KPT（ケプト）法は、ふりかえりで最も広く利用されているフレームワークです。

```
+-----------------------------------+-----------------------------------+
|         Keep（継続すること）        |       Problem（困っていること）     |
| - デイリースクラムの時間が守れた    | - テスト自動化が追いついていない    |
| - ペアプロで知識共有が進んだ        | - 仕様の確認に時間がかかった        |
+-----------------------------------+-----------------------------------+
|         Try（次に挑戦すること）                                          |
| - テストコードの作成時間をスプリント計画に含める                          |
| - プロダクトオーナーと毎日10分相談する時間を設ける                        |
+-------------------------------------------------------------------------+
```

- **Keep (よかったこと・続けたいこと)**

  うまくいったことや成果を挙げ、今後も継続する習慣として確認します。

- **Problem (問題・課題)**

  作業を進める中で発生した障害や困りごとを率直に共有します。

- **Try (次に試すこと)**

  Problemに対する解決策や、Keepをさらに伸ばすための具体的な行動計画を決定します。次回のスプリントで実行できる小さな粒度にすることが重要です。

アジャイル開発の主要な計測指標

チームの健康状態や開発の安定度を把握するために、以下の指標を活用します。

- **ベロシティ (Velocity)**

  1回のスプリントでチームが完了できた作業量（ストーリーポイント等の合計）の平均値です。見積もり精度の向上や将来予測に用いるものであり、チーム間の生産性競争に使うものではありません。

- **リードタイム (Lead Time)**

  要求が発生してから、実際に利用者に価値として届けられるまでの全所要時間です。

- **サイクルタイム (Cycle Time)**

  開発者が作業に着手してから、作業が完了するまでの時間です。

よくあるアンチパターンと対策

アジャイル開発を導入する際につまずきやすい失敗パターンと、その対処法を把握しておきます。

- **名ばかりスクラム (Zombie Scrum)**

  イベントや形だけを実施し、動くソフトウェアの完成や改善への意欲が失われている状態です。成果物の完成基準（DoD）を徹底し、利用者のフィードバックを得る機会を作ることが改善の鍵となります。

- **見積もりのノルマ化**

  ストーリーポイントを個人のノルマや達成率の評価に使ってしまう誤りです。見積もりはチーム全体の計画のための目安であり、ノルマ化するとポイントの過大申告や手抜きを招きます。

- **ドキュメント不要論という誤解**

  「アジャイルではドキュメントを書かない」という極端な解釈です。アジャイル宣言は「動くソフトウェアをより重視する」と言っているだけで、設計思想や保守に必要なドキュメントの作成を否定しているわけではありません。