

Agile2

親方 Project 編

2024-05-26 版 親方 Project 発行

はじめに

この本の目的

この本を手にとっていただき、ありがとうございます。

2024年5月

編集長 親方@親方 Project 拝

免責事項

- 本書の内容は、情報提供のみを目的としております。著者一同、正確性には留意しておりますが、正確性を保証するものではありません。この本の記載内容に基づく一切の結果について、著者、編集者とも一切の責任を負いません。
- 会社名、商品名については、一般に各社の登録商標です。TM表記等については記載しておりません。また、特定の会社、製品、案件について、不当に貶める意図はありません。
- 本書の一部あるいは全部について、無断での複写・複製はお断りします。

目次

はじめに	2
この本の目的	2
免責事項	2
第 1 章 アジャイルとは何か	9
1.1 アジャイルとはやり方ではなくあり方である	9
1.1.1 アジャイルとは	9
1.1.2 アジャイルが求められるようになった背景を爆速で理解しよう	9
1.1.3 スクラムが産声をあげる	11
1.1.4 XP が産声をあげる	11
1.1.5 Do Agile から Be Agile へ	14
1.2 アジャイルソフトウェア開発宣言(4つの価値と12の原則)	14
1.2.1 アジャイルソフトウェア開発宣言とは	14
1.2.2 アジャイル宣言の背後にある12の原則	14
1.3 国内でのアジャイルの興り	15
1.4 アジャイルといつても色々ある	17
1.4.1 SCRUM	17
1.4.2 XP(eXtreme Programming)	17
1.4.3 国産のアジャイルについても触れてみよう	18
1.5 おわりに	18
第 2 章 アジャイルで幸せになれるのか～IT 古典良書を読み解く：「初めてのアジャイル開発」編～	19
2.1 本書との出会い	19
2.2 アジャイルは誇大広告か	20
2.3 アジャイルを導入する主な理由	20
2.4 アジャイルを使って失敗する方法	21
2.5 莫邪の剣も持ち手による	22
第 3 章 アジャイルの始め方 - 続け方	24
3.1 はじめに	24
3.2 アジャイルとの出会い	24
3.3 初めてのアジャイル開発	25

3.4	地道な改善	26
3.5	学んだこと	27
3.6	その後の現場で	27
3.6.1	エピソード1：前職の現場にて	27
3.6.2	エピソード2：現職の現場にて	28
3.7	これから始める人へ	28
3.8	おわりに	29
第4章	非エンジニアの人から「アジャイルって早く作れるんでしょう？」って聞かれた時 の私の一次回答	30
4.1	「アジャイルって早く作れるんでしょう？」	30
4.1.1	「早く作れる」とは何か？	30
4.1.2	どっちのプロセスが正しいか	31
4.2	アジャイルの誤解と定着しない理由	32
4.2.1	アジャイルはいつ完成するかわからない	32
4.2.2	アジャイルがなかなか定着しない理由	32
4.3	アジャイルになるには	33
4.3.1	面倒だけどやるという覚悟	33
4.4	まとめ	33
4.5	「アジャイルって早く作れるんでしょう？」への回答	34
4.5.1	頑張って作ったものは売れて欲しいわけです	34
第5章	ぼくのアジャイル Q&A 100本ノック	35
5.1	ドキュメントに関するQ&A	35
5.2	プロダクトオーナーの役割に関するQ&A	35
5.3	見積り（時間とポイント）に関するQ&A	36
5.4	ペロシティに関するQ&A	36
5.5	スプリントに関するQ&A	37
5.6	メンバーのモチベーションに関するQ&A	37
5.7	スクラム全般に関するQ&A	38
5.8	ノックのつづき	38
第6章	アジャイルの実践	39
6.1	アイデアを出す	40
6.1.1	カスタマージャーニーマップ	40
6.1.2	リーンキャンバス	40
6.1.3	ブレインライティング	41
6.2	アイデアをかたちづくる	42
6.2.1	エレベーターピッチ	42
6.2.2	顧客インタビュー	42
6.3	スクラム	43
6.3.1	スクラムの作成物	43

6.3.2	スクラムイベントとスプリント	44
6.3.3	スプリントプランニング	44
6.3.4	デイリースクラム	45
6.3.5	スプリントレビュー	45
6.3.6	スプリントトレロスペクティブ	45
6.4	アジャイルを実践する	45
第 7 章	アジャイルではリカバリーってどうやるの？	46
7.1	アジャイルではアウトプットよりアウトカムが大事	46
7.2	スプリントの目的は「スプリントゴール」	48
7.3	やらなくても良いことを最大化せよ	48
7.4	アジャイルでリカバリーってどうやるの？	49
7.5	まとめ	49
第 8 章	経験主義に基づく計画の立て方	50
8.1	計画は何のためにするのか？	50
8.2	予実が乖離するのは計画が間違っていたから	50
8.3	計画主義による計画の立て方	51
8.4	経験主義による計画の立て方	51
8.5	見積もりの精度が上がらない？	52
第 9 章	アジャイルな案件を受託する組織職の一事例	53
9.1	アジャイルという制約	53
9.2	しかし部下はそうじゃない	54
9.3	苛烈な戦場で組織職ができるここと	54
9.3.1	誰よりも楽しむ	54
9.3.2	コンディションこそ全て	55
9.3.3	仕事に学習を組み込む	55
9.4	まとめ	55
第 10 章	関係の質を改善させるためにやってよかった 12 のこと	57
10.1	はじめに	57
10.2	チーム内の良い関係性の作り方	58
10.3	チーム外（ステークホルダー）との良い関係性の作り方	60
10.4	まとめ	63
第 11 章	不安とうまく付き合う	64
11.1	とあるチームのストーリー	64
11.2	不安感に負けてしまうチームと周りの人たち	65
11.3	リスクを早く炙り出した方がいいアジャイル	65
11.4	「デザイン」が更に不安を募らせる	66
11.5	不安感とうまく付き合うには	66
11.5.1	フレームワーク通りにやっているのに、しっくりこない時	67

11.5.2 デザインがボトルネックに感じる場合	67
11.5.3 チームの周りに不安感が漂う場合	67
11.6 終わりに	68
第 12 章 アジャイルは反復してナンボ！	69
12.1 アジャイルとはなんぞや？	69
12.2 「反復」とは何か	70
12.3 「反復」なきアジャイル	70
12.4 プロダクトバックログは「プロダクト」のバックログ	71
12.5 プロダクトバックログは ToDo リストではない	72
12.6 小さな価値とは	72
12.7 まとめ	73
第 13 章 デザインから逆算して難易度を見積もるための観点	75
13.1 はじめに	75
13.2 モバイルアプリ開発において難易度を見積もる際に重要な観点を整理する	75
13.3 iOS・Android それぞれの UI 実装観点からの重要性	77
13.4 アジャイル開発の観点から iOS/Android での類似点・相違点を押さえる例	78
13.5 なぜこの観点がアジャイル開発において重要だと考えるか？	81
13.6 まとめ	82
13.7 参考資料	83
第 14 章 インセプションデッキからはじめる アジャイル入門	84
14.1 はじめに	84
14.2 開発チームに新規着任	84
14.3 朝会の目的...？	85
14.4 インセプションデッキとの出会い	85
14.5 チーム対面キックオフと得られたもの	86
14.6 キックオフのその後	87
14.7 まとめ	87
第 15 章 独学でアジャイルを学ぶ	88
15.1 独りで学ぶという選択肢	88
15.2 独習アジャイルの進め方	88
15.2.1 準備をする	89
15.2.2 「スプリント」を過ごす	89
15.2.3 スプリントを繰り返し、進化／深化させる	90
15.3 つまり、仕事の進め方なんです	90
15.4 アジャイルを説明できるようになる	91
第 16 章 アジャイルの始め方～「書籍の探し方」～	93
16.1 はじめに	93
16.2 書籍の探し方	93

16.3	まとめ	95
第 17 章 コミュニティの探し方		97
17.1	コミュニティに参加する	97
17.1.1	コミュニティに入るメリット	97
17.1.2	デメリットはあるか?	98
17.1.3	どうやって探す?	98
17.2	コミュニティを作る	99
17.3	コミュニティの居心地をよくする	100
17.3.1	雑談する	100
17.3.2	質問する	100
17.3.3	アンチハラスメントポリシーを定める	100
17.4	まとめ	101
第 18 章 アジャイルイベントまとめ		102
18.1	各地のアジャイル関連イベント	102
18.2	良いカンファレンスに出会う、ということ	104
18.3	他にもたくさんあります	105
第 19 章 スクラムマスターの資格の選び方		106
19.1	そもそもスクラムマスターに資格は必要?	106
19.2	スクラムマスターの資格について	106
19.2.1	認定スクラムマスター (Certified ScrumMaster® /CSM®)	107
19.2.2	認定スクラムマスター (Registered Scrum Master®)	107
19.2.3	Professional Scrum Master™	107
19.3	資格の選び方	108
19.3.1	講師の考え方や理念で選ぶ	108
19.3.2	講師の得意分野で選ぶ	109
19.3.3	日本語で講師と対話できるかで選ぶ	109
19.3.4	研修方式で選ぶ	109
19.3.5	他の選び方について	110
19.4	おわりに	111
第 20 章 アジャイルを勉強した後のキャリアの 5 つのロールモデル		112
20.1	最初に	112
20.2	キャリアの 5 つのロールモデルを考えることになったきっかけ	112
20.3	アジャイルに関係するキャリアの 5 つのロールモデル	113
20.4	最後に	115
第 21 章 intro		116
第 22 章 Agile を試した		117
あとがき		118

第1章

アジャイルとは何か

山田 雄一@ditflame

1.1 アジャイルとはやり方ではなくあり方である

1.1.1 アジャイルとは

「アジャイル」という言葉が一般的になってずいぶんと経ちます。

例えば、プロジェクトマネジメントの知識体系ガイドである「プロジェクトマネジメント知識体系ガイド (PMBOK(R) ガイド)^{*1}」をみてみると、2017年に発行された第6版で「アジャイル」の内容が初めて本文に組み込まれました。さらに、2021年に発行された第7版では、「価値を提供する」事に重きを置いた、アジャイル的なプロジェクトマネジメントを行うための知識体系へと大きく変貌を遂げました。

昨今では、ソフトウェア開発やプロジェクトマネジメント以外の色々な分野に波及しており、相変わらずアツい「アジャイル」ですが、そもそも「アジャイル」はどういった経緯で必要となり一般的になつていったのでしょうか？

この章では、その思想やあり方を手っ取り早く、かつ深く理解する為に「アジャイル」が発生するに至ったソフトウェア開発手法の発展の歴史を追うことで、アジャイルが求められるようになった背景について、まずは学んでいきましょう！

1.1.2 アジャイルが求められるようになった背景を爆速で理解しよう

ソフトウェア開発の黎明期（1970年頃）は国内外を問わず、情報システム産業全体の課題として、ソフトウェア開発の生産性やソフトウェア品質が非常に悪いことが大きな問題となっていました。この問題に対しては様々な開発手法が提案されました。具体的にはウォーターフォール型開発モデルや、構造化プログラミングといった、現在の開発手法にもつながる手法が提案され、また様々なプロジェクトで実践されてきました。

ここでは「ウォーターフォール型開発モデル」の興りについて挙げます。ソフトウェア開発の生産性やソフトウェア品質が悪いという課題に対し、当時（1970年頃）のITの専門家たちが参照したのが建設分野のプロジェクト管理手法です。これは、開発プロセスとしてソフトウェア開発に

^{*1} <https://www.pmi-japan.shop/shopdetail/000000000028/>

一番近いモデルで開発を行っていると考えられたためです。建築や製造業のプロジェクト管理手法をソフトウェア開発に適用したのが、最初のソフトウェア開発手法である、ウォーターフォール型開発モデルです。

なぜこれらのプロジェクト管理手法がソフトウェア開発の管理手法として適した手法であると考えられたのでしょうか。建設分野・製造分野(特にオーダーメイド品生産)の開発管理手法は「毎回異なるものを作り、前と丸々同じものを作る事はほぼ無い」という点でソフトウェア開発手法と非常に近しかったためです。当時現役のITエンジニアだった諸先輩にこの頃の話を伺った際にも、「開発管理手法が確立していなかった当時のソフトウェア開発(およびそのプロジェクト管理手法)からすると非常に革新的だった」と伺っています。

ウォーターフォール型開発は、開発プロジェクトを複数の作業工程に分割し、次の様な運用ルールで進めるものです。

- 前工程が完了しないと次工程に進まない
- 次工程に進んだあと、前工程には決して戻らない
- 開発途中での追加や変更は原則として受け入れない

なお、このウォーターフォール型開発モデルは1998年に出た書籍でも『ソフトウェアのプロジェクトは、家建築のプロセスに似た方法で進行する。』^{*2}と記載されています。少なくとも20世紀においては、ウォーターフォール型開発モデル(とそれを発展させたモデル)が主たるソフトウェア開発管理手法であるとされていました。

またウォーターフォール型開発モデルが世に出た後、これを発展させたり応用した様々な開発モデルが考案され、実際の開発手法として運用されました。

ここで、それら開発手法の代表的なモデルを挙げてみましょう。

- Vモデル：ウォーターフォール型開発モデルのテストの部分を設計の工程に対応させたもの
- プロトタイピングモデル：ウォーターフォール型開発における、リスクと不確実性を減らすためにシステムの全部ないし一部を試作して検証し、その後システム全体を作るもの
- スパイラルモデル：計画、目標・代替案・制約の決定、代替案とリスクの評価、開発とテストの各工程を何周も繰り返しながら一周ごとに設計→開発→テストと段階を追って進めしていくことでソフトウェア開発を行うもの

このように様々な開発モデルが考案され、また実際にたくさんの開発プロジェクトでこれらの開発手法が運用され、様々なソフトウェアが開発されてきました。

しかし、これらの開発手法は概ね「開発者は顧客が要求することを最初から全て理解している」「開発者は顧客にこの先要求されることを最初から全て予測している」といった前提が成立すればうまくいく、これから変化にも耐えられるという点に基づいており、「最終の完成形のソフトウェアを作りきればその後は弄らない(弄りたくない)」という認識に立ちます。言い換えれば、仕様書が過不足なく完璧に作られている、といった、現実的には不可能なことが必要条件となっています。しかし、未来は予測できない不確実なものです。

また、これらの従来型の開発手法でも一応、状況の変化などは考慮しています。ただし、その考え方、「状況の合わない部分が出ることはあって、その際に小規模な改修はするが、基本は大きく合わなくなれば丸ごと作り直せば良い」という考えに立つものです。

^{*2} ソフトウェア工学-理論と実践, Shari Lawrence Pfleeger, 桐原書店, 2001 から引用

しかしながら、実際は、1998年に出了本でも『伝統的な「ウォーターフォール」のアプローチで開発を行うことは、今日のシステムではもはや柔軟でもふさわしくもない。』^{*3}と記載されています。ここまで挙げた各種の開発手法が、「(特に、未来の)変化に耐えられる」か?という観点では大きな課題を持っているという事実は、当時のソフトウェア開発手法の共通の課題であった事が伺えます。

そして、この課題に対する一つの答えであるアジャイル的な開発プロセスが示されるにはもう少し時間が必要だったのでした。

1.1.3 スクラムが産声をあげる

時は前後して1993年、ソフトウェア開発手法としての「スクラム」が産声を上げます。これは、1986年に野中郁次郎博士と竹内弘高博士が、当時の日本の製造業における革新的な開発手法を分析し、「スクラム」と名付けて発表した論文^{*4}が原典です。

さらに、1993年にJeff Sutherland、John Scumniotales、Jeff McKennaの3名がソフトウェア開発に適用した設計・分析ツールを構築し、また時を同じくして、Ken Schwaberも自社でのソフトウェア開発にこの手法、スクラムを用いました。

その後、これらの取り組みが1995年のOOPSLAカンファレンスで共同発表されます。この内容は後に「アジャイルソフトウェア開発スクラム」(桐原書店、2003)という書籍としてまとめられたことで、広く知られる事になります。英語版はAgile Software Development with SCRUM(Series in Agile Software Development)(Prentice Hall,2001)ですが、日本語版は絶版なようです。^{*5}

1.1.4 XPが産声をあげる

更に、ほぼ同時期に、XP(eXtreme Programming)も産声を上げます。時期がスクラムとほぼ同時期だったのは偶然ですが、筆者としてはソフトウェア開発手法の改善に対する社会的な要請が高かったからだろうと推測しています。

XPはKent Beck、Ward Cunninghamによって生み出され、その後1996年にKent BeckはRon Jeffries、Martin Fowler、Ron JeffriesとともにプロジェクトでXPの実践を行いました。

そしてこれらの実践におけるの知見などを踏まえ、1999年Kent Beckにより『Extreme Programming Explained: Embrace Change』(邦題:エクストリームプログラミング)という一冊の本^{*6*7}が世に出ました。

^{*3} ソフトウェア工学-理論と実践, Shari Lawrence Pfleeger, 桐原書店, 2001 から引用

^{*4} The New New Product Development Game <https://hbr.org/1986/01/the-new-new-product-development-game> 1986

^{*5} 一応古本であれば入手は可能です。成立の経緯などは書籍に詳しいので、興味のある方は入手して読んでみても良いかもしれません。

^{*6} 以下、XP本と記載します。邦訳の第2版は、現在も普通に購入可能です。
<https://www.ohmsha.co.jp/book/9784274217623/> また、上で記載したXPが初めて実践されたプロジェクトについての話も12章「はじまりの物語」に記載があります。

^{*7} Kent Beck, Cynthia Andres, Addison-Wesley Professional, 1999

価値・原則・プラクティス

XPは色々と革新的でした。その中でも、XP本でアジャイル開発における重要なポイントとして示されたものが「価値」「原則」「プラクティス」です。

これは、XP本では次の様に記されています。

プラクティス：プラクティスは日常的な取り組みである。

→これは、具体的なアクションを示すものです。

価値：ある状況における好き嫌いの根源にあるものだ。

→これは、何をすべきかの判断基準を示すものです。

原則：その分野に特化した活動の指針である。

→これは、価値とプラクティスをつなぐための理屈や理由を示すものです。

また具体的な例として、XP本ではこれらの「価値」「原則」「プラクティス」^{*8}がそれぞれ示されています。

- 「価値」

- コミュニケーション (Communication)
- シンプリシティ (Simplicity)
- フィードバック (Feedback)
- 勇気 (Courage)
- リスペクト (Respect)

- 「原則」

- 人間性 (Humanity)
- 経済性 (Economics)
- 相互利益 (Mutual Benefit)
- 自己相似性 (Self-Similarity)
- 改善 (Improvement)
- 多様性 (Diversity)
- ふりかえり (Reflection) ^{*9}
- 流れ (Flow)
- 機会 (Opportunity)
- 冗長性 (Redundancy)
- 失敗 (Failure)
- 品質 (Quality)
- ベイビーステップ (Baby Steps)
- 責任の引き受け (Accepted Responsibility)

- 主要プラクティス (primary practices) ^{*10}

^{*8} 基本的にすべてXP本第2版ベースで記載します。それぞれの内容については、ぜひ原典であるXP本を読んでみてください。

^{*9} ここでいう振り返り (Reflection) は、スクラムでいうところのふりかえり (Retrospective) とは異なります。リフレクションは「内省」的な意味合いで個人の成長を促すものです。一方のレトロスペクティブは「回顧」的な意味合いで過去の反省で生産性の向上や対応方法の最適化を促すものです。

^{*10} XP本では、プラクティスを主要プラクティスと導出プラクティスに分けています。『主要プラクティス (Primary

- 全員同席 (Sit Together)
- チーム全体 (Whole Team)
- 情報満載のワークスペース (informative Workspace)
- いきいきとした仕事 (Energized Work)
- ペアプログラミング (Pair Programming)
- ストーリー (Stories)
- 週次サイクル (Weekly Cycle)
- 四半期サイクル (Quarterly Cycle)
- シラフ (Slack)
- 10分ビルド (Ten-Minute Build)
- 繙続的インテグレーション (Continuous Integration)
- テストファーストプログラミング (Test-First Programming)
- インクリメンタルな設計 (Incremental Design)
- 導出プラクティス (corollary practice)
 - 本物の顧客参加 (Real Customer Involvement)
 - インクリメンタルなデプロイ (Incremental Deployment)
 - チームの継続 (Team Continuity)
 - チームの縮小 (Shrinking Teams)
 - 根本原因分析 (Root-Cause Analysis)
 - コードの共有 (Shared Code)
 - コードとテスト (Code and Tests)
 - 単一のコードベース (Single Code Base)
 - デイリーデプロイ (Daily Deployment)
 - 交渉によるスコープ契約 (Negotiated Scope Contract)
 - 利用都度課金 (Pay-Per-Use)

XP 本でも謳われていますが、本を読むだけではその専門家にはなれず、実際にやってみて、専門家のコミュニティに参加し、そして誰かにその専門分野を教えないといけません。これはアジャイルソフトウェア開発においては、アジャイルソフトウェア開発を実際にやってみて、コミュニティに参加し、更に誰かにそれを伝える事で専門家になれる という事になります。

なぜなら、「価値」がズレてしまっても駄目だし、「プラクティス」だけをなぞってもうまく行かないもので、結局それをつなぐための「原則」がしっかりしている必要があります。さらにこれら3つの「価値」「原則」「プラクティス」を一体で使わないと、うまくいかない（アジャイルとして本来ある形になれない）からです。

また、これらを繋ぐポイントとして、「アジャイルマインド」が大事だということになります。では、「アジャイルマインド」とはなにかというと、具体的な言語化はなかなか難しいのですが……

practice) は、あなたが他にやっていることは無関係に役に立つものである。すぐに改善につながり、どれからでも安全に始められる。導出プラクティス (corollary practices) は、先に主要プラクティスを習得しておかなけば難しいだろう。プラクティスを組み合わせれば増幅効果が得られるので、できるだけ早くプラクティスを追加した方が有利である。』(Extreme Programming Explained: Embrace Change 2nd Edition からの引用

1.1.5 Do Agile から Be Agile へ

そもそもアジャイルというのは、「機敏な」「敏しょうな」を示す英単語^{*11}で、形容詞です。

このため、アジャイルを導入した最初の頃などは、「意図してアジャイルを進める」「アジャイルに関するロール・アクションを行う」時期が必要な場合もありますが、本来のありようとしては、「アジャイルである」という状態を指すものです。「アジャイルをする」ではなく、「(その開発手法が) アジャイルである」という状態に到達した状態が本来の姿で正しいということになります。

最近流行りの漫画の例でいうと、「アジャイルの呼吸・當中」みたいな話ですね。意識して行うのではなく、ニュートラルにその状態を維持する必要がある、といったイメージで考えてみましょう。

1.2 アジャイルソフトウェア開発宣言 (4つの価値と12の原則)

1.2.1 アジャイルソフトウェア開発宣言とは

スクラムやXPが本が出て一般的になった後、その真意が掴み切れず「アジャイル開発」を銘打っているが実態はそうではなかった……というプロジェクトが多く発生したようです。それに対するアンサーやガイドを兼ねたものもあるのですが、2001年に「アジャイルソフトウェア開発宣言(Agile Manifesto)^{*12}」が世に出ます。

ここでは、4つの価値とそれの元となる12の原則が示されました。

アジャイルソフトウェア開発宣言

私たちは、ソフトウェア開発の実践あるいは実践を手助けをする活動を通じて、よりよい開発方法を見つけだそうとしている。この活動を通して、私たちは以下の価値に至った。

プロセスやツールよりも個人と対話を、包括的なドキュメントよりも動くソフトウェアを、契約交渉よりも顧客との協調を、計画に従うことよりも変化への対応を、

価値とする。すなわち、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。

Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas

© 2001, 上記の著者たちこの宣言は、この注意書きも含めた形で全文を含めることを条件に自由にコピーしてよい。

1.2.2 アジャイル宣言の背後にある12の原則

アジャイル開発宣言に併せて、12の原則^{*13}も公表されました。これは次のようなものです。

12の原則

*11 例えば、<https://eow.alc.co.jp/search?q=agile>

*12 アジャイルソフトウェア開発宣言 <https://agilemanifesto.org/iso/ja/manifesto.html>

*13 アジャイル宣言の背後にある原則 <https://agilemanifesto.org/iso/ja/principles.html>

私たちは以下の原則に従う：

- (1) 顧客満足を最優先し、価値のあるソフトウェアを早く継続的に提供します。
- (2) 要求の変更はたとえ開発の後期であっても歓迎します。変化を味方につけることによって、お客様の競争力を引き上げます。
- (3) 動くソフトウェアを、2-3週間から2-3ヶ月というできるだけ短い時間間隔でリリースします。
- (4) ビジネス側の人と開発者は、プロジェクトを通して日々一緒に働くなければなりません。
- (5) 意欲に満ちた人々を集めてプロジェクトを構成します。環境と支援を与え仕事が無事終わるまで彼らを信頼します。
- (6) 情報を伝えるもっとも効率的で効果的な方法はフェイス・トゥ・フェイスで話すことです。
- (7) 動くソフトウェアこそが進捗の最も重要な尺度です。
- (8) アジャイル・プロセスは持続可能な開発を促進します。一定のペースを継続的に維持でいるようにしなければなりません。
- (9) 技術的卓越性と優れた設計に対する不断の注意が機敏さを高めます。
- (10) シンプルさ（ムダなく作れる量を最大限にすること）が本質です。
- (11) 最良のアーキテクチャ・要求・設計は、自己組織的なチームから生み出されます。
- (12) チームがもっと効率を高めることができるかを定期的に振り返り、それに基づいて自分たちのやり方を最適に調整します。

前述のアジャイル開発宣言の4つの価値と12の原則を対応・紐づけると次のようになります¹⁴。

- プロセスやツールよりも個人と対話：(4)(5)(6)(11)(12)
- 包括的なドキュメントよりも動くソフトウェア：(1)(3)(7)(10)
- 契約交渉よりも顧客との協調：(1)(2)(4)(5)(6)(12)
- 計画に従うことよりも変化への対応：(2)(3)(6)(8)(9)(10)

このように、あくまで1:1で対応しているのではなく、宣言と原則が相互に作用し合っていることがわかります。

こういったアジャイルの価値と原則を大事にしつつ、現在もアジャイルは世界中で推進されています。

1.3 国内でのアジャイルの興り

最後にアジャイルに関する国内事情がどうだったのかも見てみましょう。

海外で産まれた歴史は「スクラム」→「XP」の順でしたが、日本で広まっていった順序的には実は逆で、「XP」がまず有名になり、その後「スクラム」が浸透していったという歴史があります。

XPについては、2001年頃から情報処理学会の刊行物（会誌、論文誌）で情報が出始め、2002年

¹⁴*14 ※なお、この対応は筆者の独断と偏見によるものです

3月^{*15}、4月の会誌^{*16}でXPについてのまとめた記事が掲載されたことが契機のようです。

また、スクラムについては2004年にはじめて、情報処理学会の研究報告^{*17}が上がっています。

当時の雑誌媒体についても2誌で情報を調べてみました。おおよそこの歴史を裏付ける内容で、情報が出てきたのも同様の順でした。

• XP

Software Design誌

2002年09月号の特別企画「これでわかった！XPの使い方」で一般的な記事が出た

？ 2002年8月号からエクストリームプログラミングについての記載が始め、

WebDBPress誌(2001/11/15) Vol.5 のPHPこども電話相談室 が最初^{*18}

• スクラム

Software Design誌

2005年08月号の特集記事「緊急レポート！最新ソフトウェア開発手法事情」の中で、「4章 繰り返し型開発の落とし穴 失敗から学ぶ効果的運用」が最初でした。とはいえるあまりガッツリした内容ではありませんでしたが。

WebDBPress誌(2008/5/25) vol.44

補足 Plz>ふーれむ

このように、雑誌での取り上げられた時期としてはXPの方が早いことが確認できました。日本でのアジャイルの興りは2002~3年頃に来た、XPブームがおそらく最初の大きな波であったことがわかりました。

^{*15} 平鍋健児, XP : EXtreme Programming : ソフトウェア開発プロセスの新潮流 -前編:XP概要とその周辺-, 情報処理 Vol43(2002) No3,235-241 <http://id.nii.ac.jp/1001/00064473/>

^{*16} 平鍋健児, XP : EXtreme Programming : ソフトウェア開発プロセスの新潮流 -後編:XP実践事例の紹介-, 情報処理 Vol43(2002) No4,427-434 <http://id.nii.ac.jp/1001/00064181/>

^{*17} 藤井拓, 鶴原谷雅幸, 大津尚史, 普通のプロジェクトへの適用を目指したアジャイルな開発手法の構築と適用結果, 情報処理学会研究報告ソフトウェア工学(SE),2004, Vol87(2004-SE-145),15-21, <http://id.nii.ac.jp/1001/00021293/>

^{*18} ただしUnit Test(単体テスト)の為のクラスPHPUnitを取り上げたものであり、あんまりガッツリした記事はしばらくない

XPがやって来た(黒船がやってきたのノリで)

実際、XPは革新的であった。……というよりはあまりに革新的すぎて、国内でも話題になった時は大きな衝撃を伴うものでした。

XPが話題になった当時、筆者は大手のSIerでシステム開発に従事していました。XPが話題になった時に、「言ってる事はわかるけど社内でコレ適用できんの?」という話題で盛り上がった事を今でも鮮明に覚えています。

なお、SIerが当時一般的に行っていた受諾開発とXPは基本的に食い合わせが悪く、筆者は結局その会社ではアジャイル的なプロジェクトに従事する事は無かったのでした。

結局XPは当時(2002~3年ぐらい)としてはあまりに先進的すぎたため、色々なアプローチがバズワード化してしまったり……と、それはそれで当時はネタにもされたものではありました(特にSI)

「何故か2人で1つのPC使ってコード書くらしいで」「テストから先に書くの?」「めちゃくちゃ小さい粒度でプログラム書くらしいで」「ユーザーも設計開発に参加すんの?」など。

なお、なぜこうなってしまうか……という国内的な事情としては、アメリカの企業は基本的に自社システムは社内でシステムを内製するが、日本の企業はSIer(システムインテグレーター)にシステムを外注するという業務システムに対する開発体制の大きな差があり、こういった差をXPでは埋められなかったという話はありそうだな……と思っています。

1.4 アジャイルといつても色々ある

最後に、現在一般的となっているアジャイルプロセスや、国産のアジャイルプロセスと呼べるものについて、原情報を中心に紹介しておきます。

1.4.1 SCRUM

前述の契機となった話のあと、Jeff Sutherlandはこれらを整理・構築し、2011年に「スクラムガイド」^{*19}をまとめ上げます。

最新のスクラムガイドは以下に掲載されており^{*20}、誰でも読めるようになっています。

また、邦訳版^{*21}も同様に掲載され、公開されています。

1.4.2 XP(eXtreme Programming)

XPに触れる場合、まず原情報であるXP本を参照するのが良いでしょう。

XP本は、オーム社から日本語版が「エクストリームプログラミング」(2015)として、第2版が

^{*19} https://res.cloudinary.com/mitchlacey/image/upload/v1589750939/Scrum_Guide_July_2011_i7cho9.pdf

^{*20} <https://scrumguides.org/>

^{*21} <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Japanese.pdf>

出ています。

1.4.3 国産のアジャイルについても触れてみよう

また、国産のアジャイル的なアプローチについてもいくつか紹介しておきましょう。

1つは、IPA(情報処理推進機構)が出している「SPINA3CH(スピナッチキューブ)」です。これは開発者自らがモデルベースアプローチにより自律的に改善を行うためのメソッドで、アジャイル的に開発現場で開発の課題を解決していくためのもので、現在はISO/IEC TR 29110-3-4(ソフトウェア開発における自律改善手法)として国際規格にもなっています。

PDFは無償DLできる^{*22}のでぜひ読んでみましょう。

もう1つは、サイボウズ株式会社が出しているkintone SIGNPOST^{*23}です。

概要については、ページからそのまま引用します。

「kintone SIGNPOST(キントーンサインポスト)」は「kintoneで継続的な業務改善をするための道しるべ」として、kintone経験者の考え方やコツを体系的・網羅的にまとめたコンテンツです。^{*24}

簡単に言うと、ノーコードツールを現場で継続的に改善しながら業務にフィットさせて使っていくためのコツやノウハウ、考え方をパターンとしてまとめ、共有知としたものです。kintoneでの開発・運用は上の様な形を目指すため、アジャイル開発になります。

1.5 おわりに

ここまで、アジャイルに至る歴史やその背景、興りについて学んできましたがどうだったでしょうか？

さて、次のページからは具体的な「みんなのアジャイル」の話に移っていきます。

ぜひ、いろんな章から「みんなのアジャイル」を取り込んで、自分のアジャイルに活かしていくくださいね！

ふーれむ @ditflame <https://twitter.com/ditflame>



大阪在住。最近は仕事でドキュメントを書き、趣味でドキュメントを書いています……
(- 'Д - ') キリッ

^{*22} <https://www.ipa.go.jp/archive/publish/qv6pgp00000010et-att/000027628.pdf>

^{*23} <https://kintone.cybozu.co.jp/kintone-signpost/>

^{*24} <https://kintone.cybozu.co.jp/kintone-signpost/about.html>

第2章

アジャイルで幸せになれるのか～IT古典良書を読み解く：「初めてのアジャイル開発」編～

伊藤慶紀

2.1 本書との出会い

「健康とは、できる限りゆっくりとした速度で死に向かうことしかない。」^{*1}

IT古典良書を読み解くということで、Craig Larman（クレーグ・ラーマン）の『初めてのアジャイル開発 スクラム、XP、UP、Evoで学ぶ反復開発の進め方』（日経BP,2004,以下『初めてのアジャイル開発』）を紹介します。なぜか各章の最初に名言が表示されていて、冒頭の名言は第3章 アジャイルからの引用になります。



▲図 2.1 初めてのアジャイル開発書影

当時、筆者はまだまだ若輩者のエンジニアでしたが、いくつかのプロジェクト開発を経験し、規模の差はありますが必ずといって炎上していました。原因は終盤で仕様変更が入ったり、追加

^{*1} 『初めてのアジャイル開発～スクラム、XP、UP、Evoで学ぶ反復型開発の進め方～』, 第3章「アジャイル」) より

～

漏れがあったり、大きな障害が見つかったりとさまざまでした。なんとか納期に間に合っても開発陣はボロボロ、顧客も望んでいたシステムとはズレがあってもこれでヨシとする風潮が当時の（ひょっとして、今も!?）ソフトウェア業界でした。

なんとかならないものかと考え、アジャイル開発やテスト駆動開発等に興味を持ち独学で勉強し始めます。当時、「@ IT IT アーキテクト塾 テストファーストの実践」^{*2}でパネラーから紹介されていた書籍の1つがさきほどの「初めてのアジャイル開発」で、いろいろと疑問に思っていたことがスッキリしたことを覚えています。

探したら、該当記事がありました。懐かしいですが本質は変わっていないですね。アジャイルは黎明期で、スクラムとXPを組み合わせるのが流行っていました。ペアプログラミング以外は今でも使うべきかと。また、テスターが必要とも書かれています。

2.2 アジャイルは誇大広告か

アジャイルは誇大広告という話があります。以前からある考え方（反復型など）を誇大広告して再利用しているだけなのでは？ という問い合わせ、「初めてのアジャイル開発」では「YESでもありNOでもある」と書かれています。

どういうことかというと、いわゆるアジャイルな考え方は一昔前の再利用であるため一面ではYESだが、スクラムなどの原則やプラクティスを全体としてみると新しいものなのでNOということです。

ここで学べることは、アジャイルをバズワードのように用いるといわゆる「なんちゃってアジャイル」となり、「話が違うじゃないか」とまさに誇大広告^{*3}になってしまい誰も幸せになれない結果になります。正しくアジャイルを用いることで誰しも幸せになれる可能性があがるということです。

今回はアジャイルの代表的な手法である「スクラム」に関する用語が出てきますので用語の意味を学んでおくとより分かりやすいですが、知らなくても考え方は伝わるかと思います。

2.3 アジャイルを導入する主な理由

よく聞かれる質問に「じゃあ、アジャイルにはどんなメリットがあるの？」があります。これにどう答えるのがいいのでしょうか。「第5章 導入理由」にあるアジャイル導入理由の見出しから特に大事だというものをみていきましょう。

- ・反復型開発の方がリスクが低く、ウォーターフォール型の方がリスクが高い

→ リスクが高い工程が先に来るのがアジャイル、あとに来るのがウォーターフォールとなっています。リスクグラフを見るとわかりやすいです。

^{*2} <https://www.itmedia.co.jp/im/articles/0602/24/news137.html>

^{*3} 誇大広告といえば個人的には「ビッグデータ」を思い出します。バズワード化した時期に導入して効果をあげた企業が一体何社あるのか…… また、「M2M(Machine to Machine)」がいつの間にか「IoT(Internet of Things)」と言葉を変えて領域を増やしたりと、IT業界は不思議な流行りの用語が飛び交います。



▲図 2.2 リスクグラフ：ウォーターフォール



▲図 2.3 リスクグラフ：アジャイル

・最終製品がクライアントの真の希望に適ったものになる

→ 早い段階で評価やフィードバックを繰り返すため、製品が望んだものになる可能性が高くなります。これが、いつも筆者がウォーターフォールでモヤモヤしていた「顧客が望まないものがリリースされる」ことを解決する方法になると考えています。

・タイムボックスの利点

→ アジャイルは短い期間で区切って開発をします。タイムボックスを導入するだけで生産性が上がるという利点があるそうです。いくつか理由がありますが、1つ目は「集中」。締め切りギリギリのときに驚くべき生産性を出す方も多いでしょう。締切までの時間が長いと人はだらけてしまうようです。また、タスクを現実的に対処できる程度のものに縮小し、困難な決定を早く行うようになる効果があるようです。

そして、もう一つタイムボックスの価値は人間の不思議な習性に関連することです。それは人は期限を守れなかったことはよく覚えているが、内容が少しぐらい不足していても気にはしないというものです。100% を要求するウォーターフォールと、優先順位が高い機能から作成し75% でも納期にリリースできるアジャイルを表しているようです。

2.4 アジャイルを使って失敗する方法

いざ、アジャイル開発を始めても失敗することはあります。いざ、アジャイル開発を始めても失敗することがありますが、どうすると失敗するのかを知っておけば事前に防げるかもしれません。

～

ここでは代表的なアジャイル手法であるスクラムを使って失敗する方法を取り上げた部分を見てみましょう。(第7章 スクラム)

- ・**自律的なチームでない。マネージャーまたはスクラムマスターを指揮・編成している**

→ こちらは立ち上げではとても難しいです。マネージャーは解決策を提示して指示しないといけないと思いますし、メンバーは逆に指示を仰ぎがちになってしまいます。徐々にでもいいので自律的なチームを作っていくましょう。個人的に一番よくないことが、決めたスプリントバックログを必ず終えるために、スプリント内でスコープ調整などをせずにウォータフォールのように稼働をあげて対応してしまうことです。

- ・**スプリントや個人に対して新しい作業が追加される**

→ スプリント中は要求を変更しないことが大前提です。しかし、緊急でどうしても必要な変更であるという場合は、バックロググルーミングなどを使いタスク調整・仕様調整をすることが大事だと考えます。気軽に追加・変更ができるようでは、スプリントの意味がありません。

- ・**プロダクトオーナーが参加していない、あるいは判断を下していない**

→ 他にもプロダクトオーナーが複数存在したり、最終意思決定が出来なかつたりといった問題も散見されます。個人的にはプロダクトオーナーの意識、熱量不足があるとうまく回らないことが多いようです。

- ・**ドキュメントが不十分である**

→ アジャイル開発はドキュメントを作らなくていいという話を聞いた方も多いかと思います。しかしが、反文章主義ではなく、成果物として定義していないだけであり、価値があるのであれば作成するべきです。

すでにアジャイル開発を行っている方もいると思いますが、上記のリストに思い当たる節がない方！ おめでとうございます！ 「アジャイルで幸せになれる」を体現できるはずです。逆に思い当たる節だらけの方、既に様々な歪みが起きていると思います、勇気を持って変革していきましょう。

2.5 莫邪の剣も持ち手による

「莫邪の剣（ばくやのつるぎ）も持ち手による」^{*4}という言葉がありますが、どんなに優れた名刀でも持ち手が臆病であったりすると、その真価が発揮できないという意味になります。アジャイルは確かに優れた武器ですが、使い方を誤るとその真価が発揮できません。結果が出ないからと次々とサービスや商品を乗り換える方もいますが、使う側に問題があるというのは往々にしてあることです。

結論としては「アジャイルで幸せになれるが、持ち手による」ということでしょう。

最後に、スクラムが成功する価値について、第7章 スクラムより、特に大事な箇所を引用します。なお、一部の用語を分かりやすいよう現代風に改めています。

良い持ち手になりましょう。

- ・**コミットすること**

スクラムチームは、そのスプリントの目標を達成することをコミットする代わりに、達成

^{*4} 「莫邪の剣も持ち手による」。武器は良くても使う人がダメだと効果が発揮できないという意味。「宝の持ち腐れ」ではしっくりこないのでよりよいことわざは無いかなと探したところ、ぴったりのものを見つけた次第です。筆者も初めて使いました。

するにはどうするのが一番よいかを自分たちで判断する権限と自治権が与えられる。経営陣とスクラムマスターは、スプリントに新しい作業を追加しないこと、チームに指図しないこと、リソースを提供しディリースクラムで挙げられた障害を迅速に取り除くことをコミットする。プロダクトオーナーは、プロダクトバックログを定義して、その優先順位を付け、次のスプリントの目標を選択するのに際してチームを導き、各スプリントの結果をレビューしてフィードバックすることをコミットする。

・敬意を払うこと

または責任転嫁するのではなく、チームで責任を持つこと。チームのメンバーがそれぞれの長所／短所に敬意を払い、スプリントが失敗しても誰か一人の責任にしない。マネージャーではなくチーム全体が、自己組織化と自律によって、グループで解決策を調べて「個人の」問題を解決するという姿勢をとる。また、専門のコンサルタントを雇って足りない知識を補うなどといった難問に対応するための権限とリソースを与えられる。

・勇気を出すこと

経営陣は、勇気を持って、適用型の計画や方向性を示し、メンバー個人やチームを信用してスプリントを行う方法に口出ししないようにする。チームは自主性と自己管理を必要とする仕事に勇気を持ってあたる。

アジャイルを成功させる方法は「コミットすること」に集約されていますが、最終的には、「敬意を払う」、「勇気を出す」といった、技術ではなくマインドが大事なんだなということを再認識すると共に、IT 業界に○○信者や○○教といった言葉がはびこっている理由もなんとなく分かってしました。アジャイル開発をしていない人にも敬意を払いましょう！

そして、「初めてのアジャイル開発」を執筆した Craig Larman 氏に敬意を払いたいと思います。「初めてのアジャイル開発」では、紹介した項目以外にもウォーターフォールが新規製品開発に向かない理由、アジャイルを導入する理由・失敗する方法、スクラムが成功する価値について詳細がまとめられていますので、是非手にとってみてはいかがでしょうか。



伊藤 慶紀

大手 SIer にて業務用アプリケーションの開発に従事。ウォーターフォールは何故炎上するのか疑問を感じ、アジャイルに目覚め、一時期、休職してアメリカに語学留学。Facebook の勢いを目の当たりにしたのち、帰国後、クラウド関連のサービス・プロダクト企画・立ち上げを行う。その後、ベンチャーに転職し、個人向けアプリ・Web サービスの PM、社内システム刷新など様々なプロジェクト経験を経て SHIFT に入社。趣味は将棋、ドライブ、ラーメン、花火、読書など

第3章

アジャイルの始め方 - 続け方

砂田 文宏 @orinbou

3.1 はじめに

この章は、すでにバリバリと濃いアジャイルを実践している人には当たり前のことかもしれません。むしろ、以下のような境遇の人に読んでいただき、何かヒントになるものが得られればうれしいです。

想定する対象読者（読んでほしい人）

- ・過去アジャイルに取り組んでみたけれど、うまくいかなかった人
- ・現在アジャイルに取り組んでいるけれど、うまくいっていない人
- ・これからアジャイルに取り組もうとしているけれど、どう始めたらいいかわからない人

3.2 アジャイルとの出会い

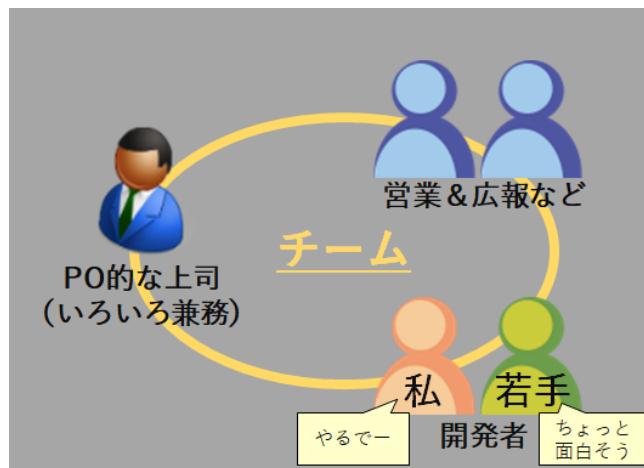
「今よりもっとうまくソフトウェアを開発したい」「価値あるソフトウェアを利用者に届けたい」ソフトウェア開発を志す人ならば、一度はそんなことを考えたことがあるのではないでしょうか？

実際にそんなことができたら本当にどんなに素晴らしいことでしょう。でも現実は厳しく、「誰が」「いつ」「どんな目的で」決めたのかよく分からぬ開発ルールという枠組みの中で、あまり役に立たないと感じながらも「でもそういう規則だから…」と、渋々あるいは必死に不合理と戦って苦しんでいるケースも未だに多いかもしれません。

そんな状況に耐えかねてソフトウェア開発から逃亡しようと画策しました。しかし、幸か不幸か2008年の秋に発生したリーマンショックで、初めての「転職」という名の逃亡が見事に失敗したあと、生きる（食いつなぐ）ために不本意ながら再びソフトウェア開発の仕事へ戻ることにしました。日々の作業に追われる中、2012年頃たまたま本屋で目にした手に取ったのが書籍『アジャイルサムライ－達人開発者への道』(オーム社、2011)でした。今から遡ること遙か10年以上も前のことです。衝撃的な内容に興奮して一気に読み終え、気になった部分を蛍光ペンでマーキングし、付箋を貼りまくったことを今でも覚えています。

3.3 初めてのアジャイル開発

当時景気が悪かったことが幸いし、アサイン先がなく社内で浮いていた人員を活用して、自社サービスのプロトタイプを開発するというプロジェクトが立ち上りました。『アジャイルサムライ』を読んでアジャイル熱に浮かされていた筆者にとって、これは千載一遇のチャンスでした。「アジャイル開発でやらせてください！」と上司に懇願して、何とか受け入れてもらいました。そして「約2週間のスプリントごとに動くソフトウェアを届けます！」と（実際できるかどうかわかりませんでしたが）約束しました。その当時のプロジェクト開始時のチームの体制は、図3.1のようなものでした。



▲図3.1 プロジェクト開始時のチームの体制

当時の私のアジャイルのインプットは、『アジャイルサムライ』とネットを漁って調べた情報だけでした。とにかくアジャイルだかスクラムだかもよく分からぬまま手探りで始めました。まず最初にやったこと（プラクティス）は次の3つでした。

- 朝会（デイリースクラム）
- スプリント（約2週間）
- ふりかえり（レトロスペクティブ）

「たったこれだけ？」と思うかもしれません、たったこれだけでも序盤からいきなり躊躇しました。まず、朝会が時間どおりに始まりません。上司に声をかけられたプロダクトオーナー（以下、「PO」と呼ぶ）が、頻繁に朝会をスキップしてきました。さらに、確かに合意したはずの「スプリント開始時のWIP制限^{*1}」を無視して、どんどんと追加タスクを積んできました。プラクティスが理解されず、乱れるリズム。つのるイライラ。駄々下がる開発者のモチベーション。

万事こんな調子で、序盤は思った以上に思ったように進みませんでした。加えて、私の相棒で

^{*1} WIP制限：進行中の作業（Work in progress）の量を制限すること。同時進行できる作業量を制限することで、タスクの切り替えに伴う無駄を省いたり、一つのタスクに集中することで早く完了することが期待できます。また、早く完了することにより、フィードバックが増えたり品質が向上することが期待できます。

ある若手開発者が「僕は過去を振り返らない男なので」と、頑なにふりかえりを拒否したりもしました。新しいことを始める場合は「たったこれだけ？」のことでも本当に大変だと感じました。

3.4 地道な改善

序盤の躊躇もあり、アジャイルごっこにもなっていない状況で何度も挫折しかけました。しかし、相棒に懇願して何とか「ふりかえり」を継続してもらい、地道な改善を続けたことでなんとか継続することができました。当時実践してみて効果を感じられた改善例をいくつかご紹介します。

・ご近所さん（プロジェクト関係者）の見える化

そもそも共通の目的に向かうワンチームとしての意識が希薄で、目先の利害が一致せず、不穏な空気になる場面が何度もありました。また、チームに影響を与えるステークホルダーの認識も人それぞれで、議論が空中戦になってしまふことも一度や二度ではありませんでした。そこで、『アジャイルサムライ』のインセプションデッキを参考に、「ご近所さんを探せ」を行い、ステークホルダーを含めたプロジェクト関係者を洗い出しました。**さらにその図を壁に貼り出して、自分たちが同じチームメンバーだとということを視覚的に認識できるようにしました。**こうすることで、チームとして共通の課題と対峙し、解決に向けて建設的な協力がしやすくなりました。また、協力しやすい空気が醸成されたことにより、スプリントの途中で開発中のソフトウェアを開発者が積極的に披露して、より素早いフィードバックを得ることも可能になりました。

・バックログの種類をプロダクト用とスプリント用に分けて管理

見様見真似で始めたこともあり、序盤はバックログの粒度もかなり混沌としていました。POや営業や広報は、ビジネス価値が関心事の中心です。一方で、開発者は、ビジネス価値を生み出すために必要な具体的な作業が関心事の中心になります。このように、観点や粒度が異なるバックログが混在し、かつ、膨大な数になっていったことで、徐々に管理が煩雑になっていきました。その結果、作業状況の確認やフォローも難しくなり、デイリースクラムの時間もどんどん長くなっていました。そこで、バックログを以下のように明確に区別して管理することにしました。

- プロダクトバックログ：

- ビジネス価値（利用者にとっての価値）を実現する単位

- スプリントバックログ：

- プロダクトバックログをスプリントで実現するため具体的な作業に細分化した単位

こうすることで、POや営業や広報は「プロダクトバックログ」だけを気にすればよくなります。また、開発者は、「プロダクトバックログ」を理解しつつ、それを実現するための具体的な作業として「スプリントバックログ」を意識的に利用できるようになりました。このように、バックログの種類を分けて管理するようにしてからは、管理の煩雑さがかなり軽減されました。また、以前よくあったPOのマイクロマネジメント（細かなタスク単位の指示）もかなり減ったように思います。

地道な改善を継続的にコツコツと続けて、その効果が実感できるようになると、チームメンバーの意識も少しづつ変わっていきました。自称「過去を振り返らない男」を明言していた

若手開発者が、ある時ふいに「環境って自分たちで変えられるものなんですね。僕、知りませんでした」と言ってくれました。この言葉を聞いて私も少し救われた気持ちになりました。それ以来、彼は社内の特別な協力者になり、社内勉強会の企画や社外コミュニティでの発表など、事あるごとに積極的に協力してくれるようになりました。その後、彼と共に成功も失敗も数多く経験することになるのですが、それまでと決定的に違ったのは、彼の協力がなかった頃と比べて、社内あるいはプロジェクトで新しいことを試すまでに要する時間が圧倒的に短くなったことでした。

3.5 学んだこと

変革を推し進めるためにはリーダー（先達）が必要だということは、誰しも認めるところだと思います。しかし、リーダー独りだけでは物事はなかなか思ったようには進みません。変革の取り組みを支持してくれる最初の協力者（以下、「ファーストフォロワー」と呼ぶ）がたった一人居てくれるだけで、新しい取り組みに対する推進力がまったく違ってきます。更に二人なら、失敗も最高の「ふりかえり」のネタ（むしろ宝物）として前向きにとらえることができるようになります、一人では躊躇して（人知れず）諦めてしまうようなケースも激減します。

このように、変革の取り組みを支持するファーストフォロワーが存在することで、最初は空飛に見えるような取り組みにも信憑性が生まれ、多くの人たちが参加するきっかけとなり、やがて大きな変革へと繋がっていくことがあります。この仕組みについて、アメリカの起業家デレク・シヴァーズ（Derek Sivers）氏が2010年のTEDカンファレンス『社会運動はどうやって起こすか（How to start a movement）』^{*2}で分かりやすく説明してくれています。まさにこれは、私がアジャイルを実践していく中で学んだことそのものでした。

3.6 その後の現場で

その後もさまざまな困難に遭遇しましたが「ファーストフォロワー」を見つけて、まず「二人」から始めることで、私なりのアジャイルな取り組みを続けることができました。

3.6.1 エピソード1：前職の現場にて

客先に常駐して、顧客側のエンジニアと自社エンジニアで混成チームを組んでGIS系アプリケーションを開発する現場でした。開発の人手が足りないということで、私は3ヶ月の期間限定の助っ人としてプロジェクトへ送り込まれたはずなのですが、何故か2年以上そこで過ごすことになりました。その現場で取り組んだエピソードを少しだけ紹介します。

Step1：若手エンジニアと「二人」で朝会を始める

派遣先の現場で若手エンジニア（顧客）と二人で朝会を始めました。毎週ふりかえりを実施して継続的な改善を続けたことで、他のチームから注目されるようになりました。最終的に当時の現場の全チーム（3チーム）に朝会とふりかえりが定着しました。

Step2：シニアエンジニアと「二人」で勉強会へ参加する

^{*2} https://www.ted.com/talks/derek_sivers_how_to_start_a_movement/ TEDカンファレンス2010『社会運動はどうやって起こすか（How to start a movement）』

アジャイルに興味を持ってくれた派遣先のシニアエンジニア（顧客）を誘って二人で社外の勉強会へ参加しました。勉強会で得た気付きや知見を糧（肴）に、あるべき姿や目指すべき方向について何度も話し合いました。一緒に様々な試みを行いましたが、例えばコミュニティのツテで社外講師を招聘して、チームやエンジニアに向けた勉強会を開催したりもしました。改善の一環で Jenkins を導入し、これまで手動だったビルドやインストーラー作成などの定型作業を自動化して、エンジニアの作業負荷やストレスを低減しました。また、人的ミスによる不具合も大幅に減らすことができました。

3.6.2 エピソード2：現職の現場にて

その後転職して現職に至るのですが、とある受託開発プロジェクトで開発リーダーを担うことになりました。若手エンジニアが中心の数名規模の小さいチームでの開発だったため、これまで培ったアジャイル開発の知見が活かせそうだなと感じ、若手エンジニアたちへそのエッセンスを伝えながら開発を進めていました。

Step1：担当プロジェクトで若手エンジニアと「二人」でスクラムを画策

アジャイル開発やスクラムの魅力や勘所など伝えたあと、スクラムに興味を持ってくれた若手エンジニアと一緒に「なんちゃってスクラム」を始めて、筆者がスクラムマスターの役割を担いました。あえて「なんちゃって」と称して最初の心理的な導入ハードルを下げて実施しました。とはいえ、まったくの骨抜きになっては元も子もないで、厳格ではないものの勘所は押さえるように注意しながら開発を進めました。

Step2：プロジェクトマネージャーと「二人」で顧客にスクラムを提案

「なんちゃってスクラム」の進め方や成果に興味を持ってくれたプロジェクトマネージャーと協力して、顧客に対してスクラムでの開発を提案することにしました。スクラムの利点や顧客に担ってもらいたい役割などを提案書にまとめて客先でプレゼンしたところ、幸運にも受け入れてもらいました。その後の開発の中では、それなりにいろいろなことがありましたが、最終的にはしっかりと顧客へ価値を届けることができたと思います。

Step3：社内でアジャイルを推進する役割を「二人」で担う

最近では社内へアジャイル開発を推進・啓蒙する役割（CoE：Center of Excellence）を担っています。かつてのアジャイルコミュニティで知り合った仲間が偶然同じ会社に（しかも驚くべきことにはほぼ同じタイミングで）入社してきたこともあり、ここ数年は二人三脚で支え合いながら活動に取り組んでいます。

3.7 これから始める人へ

正しいアジャイルやスクラムなのかどうかを気にし過ぎないでほしいと思います。

確かに「アジャイルソフトウェア開発宣言」や XP（Extreme Programming）やスクラムなどの原則や価値を理解することは重要だと思います。しかし。物事はすべてゼロイチで割り切れるほど単純ではありません。理想と現実の間でバランスを取りながら実践し学び続けていくことがより重要だと私は考えています。「そんなの本当のアジャイルじゃない」なんて、誰から言われ

ても、どうか歩みを止めないでください。

日本におけるアジャイル開発の第一人者の平鍋健児氏も、かつて「アジャイルなんて今の仕事で使えないじゃん!」と言われる境遇に陥ったことがあるそうです。そんな時「今の仕事を今よりも良くすることはできる」という言葉に救われ、アジャイルに拘り過ぎることをやめ、アジャイルの要素を使って「いきいきとした仕事がしたい」「お客様と喜び合える仕事がしたい」という気持ちに切り替えて改善や工夫を続けたそうです。私もその言葉には何度も救われてきました。このエピソードは Agile Japan 2016 の基調講演で語られていますので、良かったら下記の記事も読んでみてください(筆者が過去に執筆した記事です)。

<https://www.manaslink.com/articles/15011>

3.8 おわりに

これまでアジャイルを学び、拙いながら実践し続けてこれたのは、その時々で良き協力者に恵まれたことが最大の要因だったと考えています。与えられた境遇の中で「ファーストフォロワー」を見つけて、まず「二人」から始めることで今日までアジャイルを実践し続けることができました。運もあるかもしれません、自分の意志で前進し、試行錯誤して足搔き続けて困っていると、不思議と良いフォロワーに出会えることが多かったように思います。社内だけでなく、社外のコミュニティなどでも、助けてくれるフォロワーがきっと居ると思います。そう信じて、まずは最初のフォロワーを見つけて「二人」からアジャイルを始めてみてはいかがでしょうか。



砂田 文宏 (すなだ ふみひろ) @orinbou <https://twitter.com/orinbou>
ブログ: <https://blog.orinbou.info/>

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)

株式会社ビッグツリーテクノロジー&コンサルティング（通称：BTC）でテックリードやスクラムマスターなどを生業としています。社内へアジャイル開発を啓蒙するべく Agility CoE メンバーとしても奮闘中です。AWS (SAP、DOP) と k8s (KCNA、CKA、CKAD、CKS) チョットデキル。趣味はゆるキャンプ△とサイクリング。

第4章

非エンジニアの人から「アジャイルって早く作れるんでしょう？」って聞かれた時の私の一次回答

荒川健太郎@

4.1 「アジャイルって早く作れるんでしょう？」

筆者はソフトウェアの開発に携わっていない社内のひとも幅広く関わらせていただいているのですが、先日いわゆる「非エンジニア」な仲間との雑談中にこんなことを聞かれました。

仲間「アジャイルって早く作れるんでしょう？」

私「う、うーん…（どう返答したらいいのだろう）」

しばらく言葉に詰まった私は、

私「ちょっと10分くらい時間もらって説明していいですか？」

と前置きしてその人にも伝わるようにアジャイルについて説明しました。その内容をまとめたものが今回の記事です。

4.1.1 「早く作れる」とは何か？

「アジャイル」という言葉は本当に厄介（親しみを込めて）で、ちゃんと説明しようとするととても10分では説明できません。もっといと私自身が全てを理論立てて説明できるとは言い切れない部分もあります。

したがって今回の説明については「早く作れるんでしょう？」の問い合わせに対する答えだけに注力しました。（アジャイルとアジャイル開発は違うだとか、アジャイルはプロセスじゃないとかそういうことはひとまず横に置いておきます）

まず私はその仲間に「早く作れる」の「早く」のイメージを確認しました。

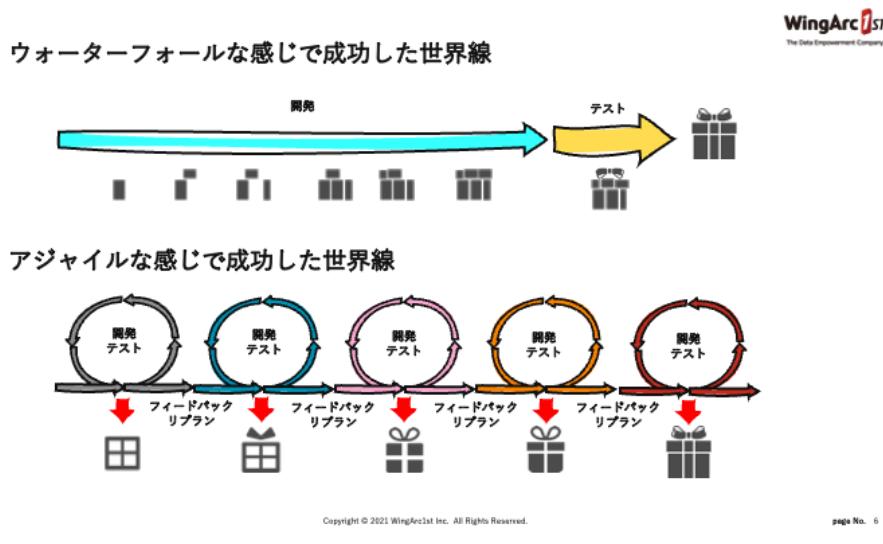
- とりあえず動く（バージョン0.1のような）状態のモノが早く完成するのか
- 企画段階で盛り込んだ機能が全て備わっている（いわゆるバージョン1.0相当な）状態のモノが早く完成するのか

仲間の回答は「後者」とのことでした。やっぱりそうかと思った私は、夏頃の新人研修で使った資料を使いながら説明を開始しました。

4.1.2 どっちのプロセスが正しいか

「ウォーターフォールとアジャイルは対となる概念ではない」ということは踏まえた上で、あえて今回はウォーターフォールと対比しています。

図4.1は、上下で違うプロセスを踏んで同じプロダクトが完成させたことを表した図です。両プロセスとも一番右のプレゼントアイコンが完成品（バージョン1.0）であると認識ください。そして仮定の話として、この完成品はとてもよく出来ていて、ビジネス的に大成功（売り上げがとても良い）と言えるクオリティのモノであるとイメージしてください。



▲図4.1 ウォーターフォールな感じで成功した世界線

さて「アジャイルが良いぞ、アジャイルが良いぞ」と言われて久しい昨今、どちらのプロセスが正しいと言えるでしょうか。

答えは、どちらも正しい。お客様喜んでるんだから。売ってるんだから。

そうです。どちらも正しいんです。

良いモノが作れるならウォーターフォールだろうがアジャイルだろうが、ここで例示していないやり方だろうがなんだっていいんです。顧客に喜んでもらって儲かることが、われわれ民間企業の目的なわけですから。

プロセスはあくまでもプロセスであって、大切なのは顧客に価値を届けることだと私は考えます。

4.2 アジャイルの誤解と定着しない理由

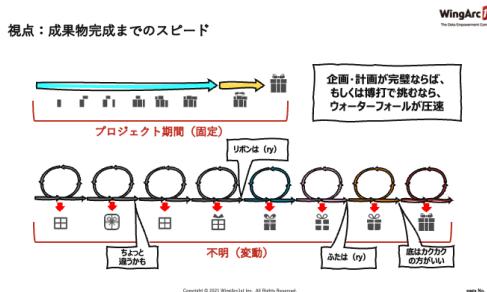
4.2.1 アジャイルはいつ完成するかわからない

一般的にアジャイルな動きには次のような利点があると言われています。

- 外部の変化に俊敏に対応できる（不確実性に強い）
- 早くから収益を見込める可能性がある
- 別プロダクトの開発の際、再現性の精度が高まる

こういった利点に注目すると「やっぱりアジャイルいいね」「アジャイルだといいモノ作れそう」となりますが、話はそんな単純ではありません。

図4.1の下部、すなわちアジャイルなプロセスの図は便宜上5つの輪っか（イテレーション＝繰り返し）でバージョン1.0が完成していますが、現実的にはもっとたくさんのイテレーションが必要となるでしょう（この図を見せた時に非エンジニア仲間から「あ、そっか！」という反応をいただいたのは気持ちが良かった）。



▲図4.2 成果物完成までのスピード

アジャイルはフィードバックを得る機会をたくさん作らないといけないため、開発に直結しない業務（プランニング、ふりかえり、レビューなどの非コーディング作業）の回数は増加し、確実にリードタイムは大きくなります。そして本当に顧客に満足してもらえるクオリティを持っていくまでの期間（輪っかの数）は未知なのです。

この時点で「アジャイルって早く（完成形を）作れるんでしょ？」という仲間の問い合わせに対する答えを伝えることが出来ました。

答えは、「アジャイルはいつ完成するかわからない」です。」

これで当初の目的は達成できたわけですが、興が乗った私はもう少し話を続けました。

4.2.2 アジャイルがなかなか定着しない理由

前項をまとめると、アジャイルは「良いモノを作れる確率を上げられるかもしれない」「早期から収益が見込めるかもしれない」けれども「開発業務以外の仕事が多くて」「開発スピードは上がらないであろう」シロモノであるということです。

私は日本の組織でアジャイルがいまいち定着しない理由は、プロセスの方法論ばかり注目され

て、それ以前のアジャイルマインドの醸成の重要性が軽視されているからだと考えています。

- 自分の業務以外のタスクが多くて忙しくてなんかジレンマ
- あまり開発に集中できない
- 結果、開発スピードが上がらない

このように、アジャイルに悩む声を筆者はたくさん聞いてきましたし、筆者自身非常に共感しています。

4.3 アジャイルになるには

4.3.1 面倒だけどやるという覚悟

ここまで説明したように、「良いものが作れるかもしれない VS. 面倒くさい」

というなかで、後者のリスクを考慮してアジャイルを選択できない組織が多い中、それでも前者を重んじアジャイルを採用したい場合どうすれば良いのでしょうか。

まずは関係者全員が「アジャイルマニフェスト」に書かれていて内容を理解し、自分たちが作ろうとしているプロダクトにはアジャイルが必要だと納得している状態になる必要があると思います。

繰り返しになりますが、アジャイルは「短いイテレーションで少しづつ作る」「決められた行事（プランニング、ふりかえり、レビュー）をこまめにやる」といった「方法」ではありません。アジャイルな開発を目指すには「アジャイルマニフェスト」とそこで謳われている価値や主義に基づいた12の原則を関係者全員が理解し実践する「状態」になる事を目指すという組織としての覚悟が必要となってきます。

こういったことからアジャイル界隈では有名な、「Don't just do "Agile", be "Agile".」（アジャイルするな。アジャイルになれ）という言葉があるんですね。関係者全員が心技体でいう「心」の部分を理解しなければ、アジャイルは成り立たないです。

ですから、「それアジャイルで作る必要あるの？」と言った自身への問いかけもとても重要だと筆者は考えます。作るモノ（完成形）は明らか、外部環境からの影響もない、それほどフィードバックも必要ない。そのようなモノは当初の企画通りに集中して作ってしまった方が、スピード、コスト、品質面でも成功的の確度は高く、アジャイルが出る幕はありません。

4.4 まとめ

まとめます。

- 「Don't just do "Agile", be "Agile".」を関係者全員が理解する
- 「アジャイルは早く作れないし面倒くさいけど良いモノを作りたいからやる」という覚悟を関係者全員が持つ
- そのプロダクトの開発はアジャイルに進めた方が効果がある（ビジネスで勝てる）ことを関係者全員が納得している

この条件を満たして初めて組織はアジャイルなプロセスへの一步を踏み出せるんだと思います。

4.5 「アジャイルって早く作れるんでしょ？」への回答

仲間「なるほどですねー。全然思ってたのと違った。すごくよくわかった。」

ここまで話をして、最初の質問への回答と反応に戻ります。仲間からの反応が嬉しかったです。

もっとも、この話はかなりデフォルメされていて多角的な視点にも欠けており、まだまだアジャイルの入り口にすぎないのですが……より詳細にはまた別の機会にお話しできればと思います。

4.5.1 頑張って作ったものは売れて欲しいわけです

最後に筆者の立ち位置としては、

「良いモノが作れるならね…最初は面倒かもしれないけどね…それぞれのプラクティスにはちゃんと理由はあるわけだから…見方とかマインドとか変化させて…みんなアジャイルになろうよ！」って感じです。

だって、

「良いものが作れるかもしれない < 面倒くさい」だからアジャイル嫌だ

こんなの勿体無いじゃないですか。日々頑張って作ったのに「お客様喜ばない&売れない」っていう事態は避けたいですよね。

「良いものが作れるかもしれない > 面倒くさい」だからアジャイルになってみよっかな！

こうあるべきだと思います。みんなの頑張りは報われるべきです、絶対。



荒川健太郎 @

ウイングアーク1stでプロセス改善とソフトウェアテストの自動化に取り組んでます
/ Certified ScrumMaster / Certified ScrumDeveloper / JBA公認E級審判 / 全日本スノーダイビング協会会長

第5章

ぼくのアジャイル Q&A 100 本ノック

木下 史彦

アジャイルコーチが「アジャイル」に関連する質問にひたすら答えていきます。

5.1 ドキュメントに関する Q&A

Q: アジャイル開発であってもドキュメントは作成すべきでしょうか？

A: ドキュメントとひとことに言ってもさまざまな種類のドキュメントがあります。手はじめに現状作成しているドキュメントを以下の3つに分類してみましょう。

1. 引き継ぎのためのドキュメント
2. 自己防衛のためのドキュメント
3. コンプライアンスのためのドキュメント

ウォーターフォール開発のように開発のフェーズによってチームを分けないのであれば引き継ぎのためのドキュメントは必要ないでしょう。

チームが信頼関係をもって仕事をできるなら、合意事項を事細かに文書化し訴訟リスクに対応するような、自己防衛のためのドキュメントも必要ないでしょう。そんなドキュメントを作っている時間があったら信頼関係を築くことに力を注いだ方が得です。

法令によって定められたドキュメントは作らないわけにはいきませんので、プロダクトバックログに積んでおきましょう。

これ以外にも重要なドキュメントはあります。ドキュメントは Word や Excel で書かれた形式的なドキュメントに限りません。たとえば、コミットログや Pull Request のコメントも立派なドキュメントです。これらには「なぜ変更したのか」あるいは「なぜしなかったのか」を記載しておくことで、ソフトウェアのメンテナンスが格段にやりやすくなります。

5.2 プロダクトオーナーの役割に関する Q&A

Q: PO が忙しすぎて役割をまっとうすることができません。どのような対策が考えられますか？

A: まず PO の役割をいくつかに分割してみましょう。PO の役割（ロール）はプロダクトの価値を最大化することですが、これを実現するために『スクラムガイド』にはいくつかの仕事（ジョ

ブ) が例として示されています。

1. ビジョン・ゴールの伝達
2. PBI の作成
3. PBI の優先順位づけ
4. PBL の見える化
5. ステークホルダーマネジメント

その上でこれらの仕事を「PO に絶対にやってもらいたいこと」と「PO 以外のチーム全体で支援するできること」の 2 つに分けましょう。

例えば、ビジョン・ゴールがしっかりと伝わっていれば PBI の作成や PBI の優先順位づけは開発者がくるかもしれません。PBI の細かな受入条件については開発者が作成して、PO が確認するというようにしてもよいでしょう。

『ビジョナリー・カンパニー ZERO』には OPUR (One Person Ultimately Responsible : 最終責任を負う者) を任命すべきだと書かれています。PO はまさに OPUR だと言えます。一方で、OPUR と同じぐらい重要なのが、誰もが隣人のために「歩道を雪かきする」文化があることだと同書には書かれています。自分の仕事に対して完全な責任を引き受けつつ、お互いの歩道の雪かきを助ける文化が大切だということです。

5.3 見積り（時間とポイント）に関する Q&A

Q: 時間の見積りとポイントの見積りの違いがよく分かりません。相対的に見積もったとしても結局、時間に換算するのではないですか？

A: スクラムで見積りをするのはベロシティを計測することが目的です。

少し長くなりますが、実例で説明します。

たとえば、ある仕事（仕事 A）があったときにそれを「3 ポイント」と見積もり、その仕事をこなすのに結果的に 6 時間かかったとします。1 ヶ月後、仕事 A と同じような仕事 B をやることになりました。仕事 A をやったことでコツが分かっていたので、2 時間でできそうだと思ったとします。ポイントも 3 分の 1 に減らして「1 ポイント」にしました。これをやってしまうとベロシティはどうなるでしょうか？6 時間でこなせる仕事は「3 ポイント」分のまま変わりません。

仕事 B をあくまでも規模を基準に「3 ポイント」と見積もったならどうなるでしょうか？3 ポイントを 2 時間でこなすことができるのであれば、6 時間あれば 9 ポイント分の仕事ができることがあります。

前者（時間で見積もる）の問題点はベロシティの変化が分からなくなるという点です（時間で見積もっているのでベロシティは常に一定）。後者（規模で見積もる）ですとチームの改善が進んでベロシティが上がっているのか、それとも何かチームに課題があってベロシティが下がっているのかが分かります。

5.4 ベロシティに関する Q&A

Q: ベロシティの考え方として、スプリント中に終わらなかった PBI は 0 としてベロシティにカウントしませんが、次のスプリントで終わらせた場合、そのストーリーのポイントを丸々カウンツするのでしょうか？その場合、本当は 2 スプリントかけて終わらせた PBI が 1 スプリントの

成果として計上されるのでベロシティが極端に上下しませんか？

A: はい。上下して OK です。

スプリントで終わらなかったというのは異常な状態ですのでベロシティが極端に上下することによって、その異常状態が見える化されます。

スクラムは問題を明るみに出すためのフレームワークです。問題が分かるのはいいことです。もしも、問題があるのに問題がないかのように取り繕おうとするなら、最初からスクラムを使わない方がいいでしょう。

5.5 スプリントに関する Q&A

Q: 1週間のスプリントでは「出荷可能なインクリメント」を作成することができません。スプリント期間を2週間に延ばしてもよいものでしょうか？

A: そういう理由でスプリント期間を延ばすことはおすすめしません。

スプリント期間中に出荷可能なインクリメントを作れないということはよくあります。しかし、それをよしとはせず「どうすれば1スプリントで出荷可能なインクリメントができるか」を考える改善のきっかけにしていくことが重要です。

改善のポイント

- 開発スキルの向上
- スウォーミング（全員でひとつの PBI に集中）
- PBI をより細かく分割（抽象化能力、設計力、本質を見抜く力が必要です）

スクラムは問題を明るみに出すためのフレームワークです。スプリント期間を短くすることにより多くの問題を明るみに出すことができます。スプリント期間を延ばすことによって、せっかく明るみに出した問題を覆い隠してしまってはいけません。

5.6 メンバーのモチベーションに関する Q&A

Q: 私はスクラムマスターです。スクラムイベントに積極的でないメンバーがいるのですが、どう促すと良いでしょうか？

A: 積極的になれない理由によって対応が変わってくると思います。

イベント趣旨や参加の仕方を単に知らないのであれば

→スクラムやアジャイル開発に関するレクチャを行う

心理的安全性が確保されていないで参加しづらいのであれば

→ワーキングアグリーメントをメンバー全員でつくる

特定の人が話しすぎることで会話に参加しづらいのであれば

→ワーキングアグリーメントを決める

→話しそうな人に対して個別にコーチングする

グループワークが苦手で会話に参加しづらいのであれば

→ファシリテーションの工夫によって均等に話せるようにする

例) 考える時間と発言の時間を分ける

 チェックイン・チェックアウトで参加のハードルを下げる

スクラムが嫌い、もしくはスクラムに疑いの目を持っているのであれば

→スクラムマスターが困りごとを個別に1on1を行い、信頼関係づくりからはじめる（スクラムをやってもらう前に、その人の関心事や困りごとを聞いて、抱えている問題を一緒に解決するような動きをするとうまくいく場合が多い）

SMとして積極的でないメンバーの内面状況が全く予想できないのであれば

→スクラムマスターが困りごとを個別に1on1を行い、信頼関係づくりからはじめる

5.7 スクラム全般に関する Q&A

Q: 何ができていたら、自分たちはスクラムができていることになりますか？

A: 逆に質問です。何ができていたら、料理ができることになりますか？

インスタントカレーが作れたらOK？ フランス料理のフルコースが作れる必要がある？ それとも、独自のレシピを発明する？ 調理師の資格は必要？ 料理に限らず、英語の学習、スポーツ、などなど・・・。

何でも同じです。スクラムも同じことです。スクラムができていることを目標にしてしまうといつまで経っても自信を持って「できています」と言えるようにはならないでしょう。目的にあわせて目指すべき達成度（目標）があるはずです。目的にあわせて目標を自分たちで考えて決めていればよいと思います。

5.8 ノックのつづき

紙面の都合で7本しかノックできませんでした。

Agile Studioでは視聴者の方から寄せられたアジャイルの悩みに、アジャイルコーチも一緒に悩みつつも解決策考えていく「アジャイルカフェ@オンライン」というイベントを定期開催しています。

<https://www.agile-studio.jp/post/agile-cafe>

続きはこちらでお待ちしています。



木下 史彦 @fkino <https://twitter.com/fkino>

永和システムマネジメント Agile Studio アジャイルコーチ

居飛車党、ニコン党、日本酒党。純米酒をあたためて呑むのが好きです。

第6章

アジャイルの実践

amixedcolor/エイミ(保 龍児)

この章では、アジャイルの実践として、アジャイルであるために用いられるいくつかのフレームワークや手法を説明します。決してこれらがアジャイルにおける全てではありませんが、アジャイルを実践する上でそれをイメージする助けになることを目的としています。また、陥りやすいポイントや重要なポイントにおいて必要に応じて補足し、本書で得たキーワードをもとに実践できる姿勢を整えます。第一歩というよりは、歩き始める前に「よーいドン」の「よーい」をしていただけるように執筆しております。

この章をどう使うかについては、もし読んでみてわからない単語や説明が多ければ、各節のタイトルを覚えていただき、アジャイルについて経験し学びを進めていく中で、その節の名前が出てきたときに立ち返ってみてください。何が重要だと書かれていたのかを改めて読み、実践に繋げることができるでしょう。そうではなく、なんとなく理解して読み進めることができたり、すでに実践していることがあれば、自分が想像していたものと何か違うところはないか、実践している中で書かれているものと違うところはないか、自分の理解と異なるところを意識して読んでみてください。この説明があなたの状況に必ずフィットするわけではありませんが、筆者の考えるポイントや概要と異なるところがあれば、その点について一考し、改善できる余地があるかもしれません。

この章のまえがきとして最後に、アジャイルを実践する上でのマインドについて記します。そのマインドとは、アジャイルを実践する方法もアジャイルに改善していくというものです。型に則るだけでなく、自分たちに合った型を探していくこともアジャイルに行ってみてください。筆者は筆者なりの考えを持って執筆しておりますが、アジャイルを実践する方法は多くあり、同じ名称でも説明やポイントが異なることがあります。アジャイルを実践する上で、知った方法をやってみるだけでなく、ぜひその方法を改善できないか考えてみてください。改善する上では、その方法が目的とするものや、実現したいことに立ち返ってみてください。実はチームでやってみているときに、型に則ることが最適ではないかもしれません。逆に、チームでやってみていることが型から外れすぎて、目的を達成しにくくなっているかもしれません。チームにとって、価値のあるプロダクトを提供することのできる、よりよい方法で実践できるように、アジャイルに振り返り調整してみてください。

6.1 アイデアを出す

プロダクト開発をする上で、まず初めには必ずアイデアから始まります。アジャイルを実践する上で、第一歩となるアイデアの出す方法について筆者の経験したいくつかを紹介します。

6.1.1 カスタマージャーニーマップ

カスタマージャーニーマップとは、一言で表した困りごとに対し、ペルソナを定めて、各フェーズごとに行動、関わるものや場所、思考や感情なども併せて書いていくというものです。困りごとを解決したいという前提がある中で、ユーザーの解決前の体験に焦点を当てて書き、イメージしやすくすることで「本当に課題があるのはどの部分の体験なのか」の目星をつけます。

このときのポイントが2つあります。1つ目は、最初に一言で表現した困りごとは、必ずしもユーザーの解決したいことの本質ではないということ。2つ目は、困りごとを解決する手段を前提にしないことです。前者については、カスタマージャーニーマップを客観的に書くことで、実は困りごとだと思っている「ここ」が、実は別の時系列にあるものが本質で、そこから派生した困りごとであるケースがあります。後者については、手段ありきのカスタマージャーニーマップになってしまふことで、困り事の本質に気づかず、今ある説を補強するだけになってしまうことがあります。この2つのポイントを押さえてカスタマージャーニーマップを書いていくことが、良質なアイデアを出す基盤になります。

とはいっても、特に2つ目にあるように手段ありきで考えてしまいやすいことが多いです。そんな時に便利な、The Job Story Format というものがあります。まずはカスタマージャーニーマップを一通り書いてから、困っているようなポイントを複数挙げます。それぞれのポイントにおいて、①それはどんな場面で、②ユーザーは何をしたいのか、③それをしたいのは何ができるからなのか、を書くことでユーザーが達成したい仕事を明確にします。困りごととその解決策への先入観のある状態から少し離れて、ユーザーが特定の場面では何をしたいのか、何ができるからそれをしたいのかと、具体的かつ客観的にユーザーのニーズを見極めます。

6.1.2 リーンキャンバス

リーンキャンバスとは、フォーマットに沿ってアイデアを書き出しながらそのアイデアを検証するというものです。すでに何かしらのアイデアがある前提で、そのアイデアをブラッシュアップしていくことに用いられます。その過程でアイデアが大きく転換する可能性もあります。



▲図 6.1 リーンキャンバスのフォーマット

<https://www.utokyo-ipc.co.jp/column/lean-canvas/>より画像を拝借。書き直しお願いします。

これにはいくつかの題目に沿った枠が用意されており、それを埋めていくことで活用します。1番目に顧客セグメントとアーリーアダプター、2番目に課題と既存の代替品、3番目に独自の価値提案を埋めていきます。誰の何をどんな独自性を持って解決したいのかを明確にし、ビジネス面におけるいくつかの事項を考慮に入れます。このとき、ソリューションを何にするかが1つの肝になりますが、ソリューションは次節.1.3で紹介するブレインライティングを活用してから定めることがおすすめの手段です。

リーンキャンバスについて短く語ることは困難ですが、初めて使用する際はぜひ枠を埋める順番に注意してみてください。枠が多いためどこから埋めたらいいかわからないといった疑問に対して、本書で書かれている埋め方で埋めることで埋めやすくなるでしょう。

6.1.3 ブレインライティング

ブレインライティングとは、フォーマットに沿った用紙をチーム内で回し、前の人と被らないようにしつつ、特定のテーマに沿って発想を派生させたり逆転させたりしてアイデアを出すものです。短い時間制限を設けた上で行い、1ターンの間にできる限り必ず全ての枠を埋めていき、そのアイデアを捻り出す過程でアイデアが柔軟なものになっていきます。

ブレインストーミングと同様、どんなアイデアでも歓迎され、突拍子のないアイデアからヒントを見つけられる可能性を大きくします。6.1.2で紹介したリーンキャンバスにおいて、今ある課題をテーマにし、そのソリューションについてブレインライティングをすることも可能です。

6.2 アイデアをかたちづくる

アイデアを出しただけでは、実際の開発に進むのは困難です。出したアイデアをかたちづくり、磨き上げ、開発に移る準備をしましょう。

もちろん、開発を始めてからもアイデアを検査する必要があります。いわゆるピボットをして、アイデアの方向性を変えることもあるでしょう。この節では、アイデアをかたちづくる2つの方法を紹介します。

6.2.1 エレベーターピッチ

エレベーターピッチとは、エレベーター内の短時間で上司や役員にアピールしても相手に必要な情報を伝えることのできる、1分程度の短いピッチのことです。様々出したアイデアを集約し、シンプルに要点をわかりやすく詰めることで、聴衆がそのアイデアの概要を短時間で把握することができるようになります。

フォーマットを調べると、表現にブレがあり一見するとどのフォーマットが良いかわかりにくいですが、一貫して次の7つを含むことが多いです。

1. 顧客の望むニーズや課題
2. 対象顧客
3. プロジェクトやプロダクトの名称
4. カテゴリやジャンル
5. 重要な利点と対価を払う説得力のある理由
6. 競合
7. 競合との違い

//noindent

これらをつなぐと、次のような文章になります。「①を望む②向けの③は、④です。これは⑤ができ、⑥とは違って⑦があります。」という文章です。

6.2.2 顧客インタビュー

顧客インタビューは、その名の通り対象顧客へのインタビューのことです。アイデアをかたちづくる上で、客観的な声を取り入れ、必要な修正を加えながらアイデアを成長させます。

顧客インタビューでのポイントは大きく2つです。1つ目は意見を誘導しないことで、例えば「こういう機能どうですか?」ではなく、「どんな機能が欲しいですか?」と聞くことです。2つ目は得た意見をただ全て取り入れるのではなく、チームでこれは確かにそうだと思ったものや、ユーザーに共通していた意見などを中心に取り入れていくことです。そうすることで素直な意見を集めながら、それでいて意見を全部取り入れて凡庸になることなく進められます。

6.3 スクラム

スクラムは、チームが価値を生み出すための軽量級フレームワークのことです。アジャイルと同様に価値基準が存在し、スクラムの理論に基づいています。アジャイルであるためによく採用されるフレームワークです。「アジャイルマニフェスト」のように「スクラムガイド」というガイドがあり、初版から数年おきに更新を繰り返しています。ガイドだけでなく、研修・認定資格・多数の本が世に出ています。

この節では、スクラムガイドにおいて定義されている中でも 2 つ、作成物とイベントについてのみ触れます。スクラムの定義・スクラムの理論・価値基準・スクラムチームについてはほとんど触れておりません。アジャイルの実践として、6.1・6.2 で扱ったように、実戦における具体的なことを中心に扱います。

6.3.1 スクラムの作成物

スクラムでは 3 つの作成物が定義されています。

1. プロダクトバックログ
2. スプリントバックログ
3. インクリメント

//noindent これらの作成物を作成することを通して、プロダクト開発を進めます。この節では、作成物について説明しながら、付随する概念についても触れます。

1 つ目の作成物、プロダクトバックログ（以降、PBL）とは、プロダクトと 1 対 1 対応で存在し、一意なプロダクトバックログアイテムの優先順位を持っているリストのことです。プロダクトバックログアイテム（以降、PBI）は、PBL を構成する要素のこと、プロダクトの改善に必要なものです。最初期には、プロダクトとして何もない（＝何も価値がない）状態から、「プロダクトとして価値を生むために」必要なものが PBI になります。PBL はプロダクトゴールをもち、プロダクトの将来の状態として、計画におけるターゲットになります。

PBL において特に重要なのは PBI の粒度です。よくない状態として、PBL にある全ての PBI が粗い、もしくは全て細かい状態があります。理想的なのは、上は細かく、下に行けば行くほどだんだん粗くなっている状態です。なぜなら、まだ見通しが聞きやすい直近の PBI を細かい粒度で分割するのは適切でも、見通しが立たないようなまだまだ後に行う PBI を細かくしても、無駄になることが多いからです。また、必要以上に細かくすることは、コストを要する上に柔軟性を失い易くなります。また、直近の PBI を適切な粒度にすることは「Ready な状態に置く」と言います。Ready な状態とは、誰が見ても迷いなくタスクに分解でき、認識に齟齬も起きない程度の粒度のことです。これにより、その PBI に期待するプロダクトの価値を実現することをより確実にします。

PBL は定期的に、また隨時、メンテナンスされることが望ましいです。そのメンテナンスのことをプロダクトバックログリファインメントと呼びます。リファインメントでは、着手が近い PBI を Ready な状態にしたり、PBL の中で PBI の位置を変更し、優先順位を調整します。

2 つ目の作成物、スプリントバックログとは、スプリント中の全てを集約するもののことです。のちに説明するスプリントプランニングで作成するスプリントゴールや、先ほどの PBL にある

PBI とそれを分解したタスク、そのタスクの状態も、ここに記録されます。

3つ目の作成物、インクリメントとは、1つのPBIのタスクが全てDONEになり、POによって受け入れられた後のPBIのことです。プロダクトの改善における踏み石とも呼ばれ、この踏み石を1つ1つ増やし、踏んでいくことでプロダクトがより価値を生むことが期待されます。

6.3.2 スクラムイベントとスプリント

スクラムでは5つのイベントも定義されています。

1. スプリント
2. スプリントプランニング
3. デイリースクラム
4. スプリントレビュー
5. スプリントレトロスペクティブ

//noindent

スクラムイベントは、スクラムの作成物の検査と適応をするための公式の機会であり、このイベントを通して作成物を検査し、状況に適応します。スクラムイベントは、複雑さを低減するために、同じタイミングと時間で開催されることが望ましいです。

1つ目のイベント、スプリントとは、スクラムイベントの入れ物となるイベントのことです。これは期間でもあり、定期的なリズムを作るのに用いられます。多くのプロダクト開発では2週間としてスプリントを用意しますが、1ヶ月以内の決まった長さであれば問題ありません。期間を短くすることで、複雑さを低減し、できる限り予測可能な範囲で計画を立てて実行し振り返ります。

6.3.3 スプリントプランニング

2つ目のイベント、スプリントプランニングとは、スプリントにおける一番最初のイベントであり、そのスプリントにおける計画を行うイベントのことです。まず初めにスプリントゴールを策定し、それに応じて必要なPBIを選択します。

望ましくないパターンが2つあります。① PBIのいくつかを先に選択することが決まっていて、それをまとめたようなスプリントゴールになることと、②スプリントプランニングの時間を超過して、計画する時間を延長してしまうことです。①については、PBIがPBLの中で適切な順序に並んでいることを前提としてしまっていて、本来はそうではなく、前回のスプリントの振り返りも踏まえた上で、「今この瞬間、次のスプリントでは何を達成すべきなのか」というものをゴールとして設定することが望ましいです。②については、計画が時間内に終わらないのであればそもそもPBIがReadyに状態になっていないことや、あまりにも不確実性が高く計画しきれないため、早く動いたほうがいい状況があります。

この2パターンはどちらも、不確実性の高い中でアジャイルにプロダクトを開発していくために避けたいパターンです。①に陥ると、逐次状況が変わるような不確実性の高い中、前回のPBLリファインメント結果による先入観が入ってしまいます。スプリントプランニングは新しい1日の頭に置かれることが多いですが、それは一時的な先入観から逃れ、できるだけ最適な選択を取るためにあります。②に陥ると、計画できる状況にない中で、最も計画を手助けする手段と言える、その計画を実施することを妨げてしまいます。また、そのような状況下で計画する時間を長く

とっても、状況は好転することが一向にないことが多いです。

6.3.4 デイリースクラム

3つ目のイベント、デイリースクラムとは、毎日決まった時間に行われ、スプリントプランニングで計画したタスクを調整しながら、スプリントゴールに対する進捗を検査し、必要に応じてスプリントバックログを適応させるイベントのことです。具体的には例えば、昨日はスプリントゴールのために何を達成したのか、今日は何をスプリントゴールのために貢献する予定なのか、その達成の障害になっているものはないか、確認します。

6.3.5 スプリントレビュー

4つ目のイベント、スプリントレビューとは、インクリメントをデモして、さまざまな人からレビューを受ける場であるイベントのことです。このイベントの目的は、スプリントの成果をレビューを通して検査し、今後どう適応するかを決定することです。さまざまな意見が出たときは、顧客インタビューのように、適切にそれらを受け止めてからPBLに反映させます。

6.3.6 スプリントトレロスペクティブ

5つ目のイベント、スプリントトレロスペクティブとは、主にスクラムのプロセスとプロダクトに対して改善する方法を計画するイベントのことです。これはよく振り返りとしてイメージされますが、振り返りを通して次の意思決定や行動を改善することが重要です。実際に撮られる手法はとても多く、例えば、アジャイルマニフェストの4つの価値を指標として、その重なりを15象限のペン図にし、プラスとデルタについて記載するような方法もあります。一般的によく用いられているのはKPTやFun Done Learnです。

6.4 アジャイルを実践する

本書の冒頭で触れたように、アジャイルとはやり方ではなくあり方です。この章では、そのあり方を実践するための方法について紹介してきました。再三になりますが、決してこれらの方法だけが全てでなければ、紹介した方法そのものも記載した説明だけにとどまりません。しかしながらこれらは、アジャイルの実践として鍵となることの多い方法です。特に、最後に紹介したスクラムは、プロダクト開発をアジャイルに行う上で非常に有用なフレームワークです。ぜひ、本書のキーワードをもとに「よーいドン」の「よーい」をして、「ドン」といつでも一歩目を踏み出せるようにしてみてください。アジャイルを実践する者として、共に歩む日を楽しみにしております。



amixedcolor/エイミ（保 龍児）

<https://twitter.com/amixedcolor>

株式会社 Relic エンジニア

新規事業開発、アジャイル、AWS、完全没入型仮想現実、歌、漫画が好きです。

第7章

アジャイルではリカバリーってどうやるの？

松永 広明

ある日の研修で、受講生のひとりからこんな質問をいただきました。

「アジャイルでは、遅れが出た場合のリカバリーはどうやるのでしょうか？」

うーん、実によい、趣のある深い質問です。実に深い。松崎しげ●の目尻のしわのように深いです（事実無根）。

この問題を理解するには、いくつかのステップが必要です。まずはそれを挙げてみます。

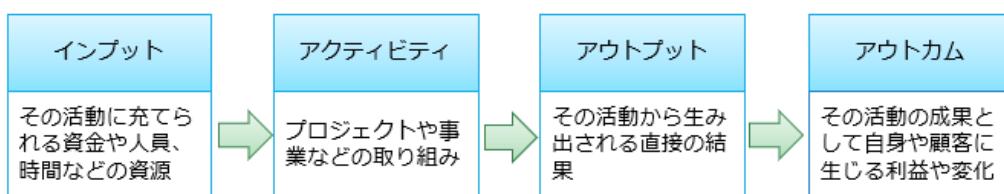
1. アジャイルは価値が大事。アウトプットよりアウトカム
2. スプリントの目的は「スプリントゴール」
3. 価値をできるだけ毀損せず、やらなくとも良いことを最大化せよ

順に解説します。

7.1 アジャイルではアウトプットよりアウトカムが大事

「アウトプット」は、プロジェクトなどの活動から生み出される直接の結果や成果物のことです。一方で「アウトカム」は、その活動の成果として、自身や顧客に生じる利益や変化のことです。

図にすると、図 7.1 のようなイメージです。



▲図 7.1 ロジックモデル

ソフトウェア開発で言えば、インプットはお金だったり開発者たちの工数だったりします。ア

アウトプットはアクティビティ、すなわち日々の開発作業から作られたソフトウェアシステムそのものです。アウトカムは、そのシステムを使うことで得られる利益や、削減できるコストや、生活の変化だったりするわけです。

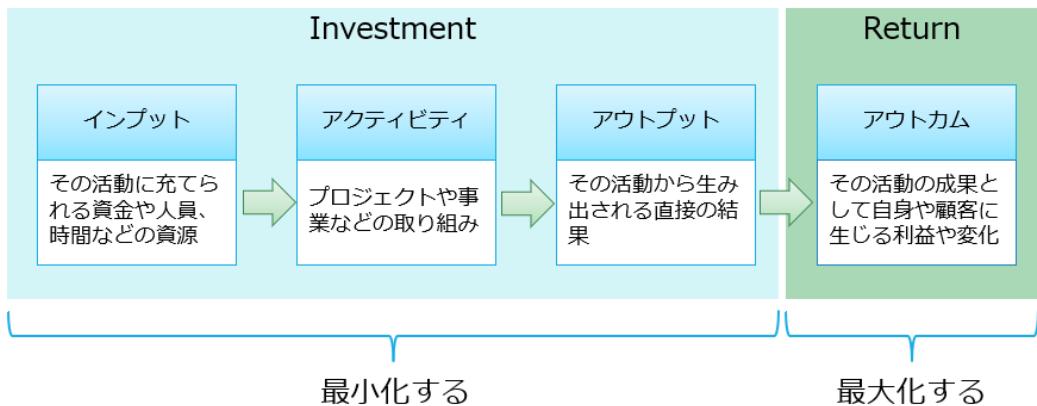
Return On Investment (ROI) という言葉があります。日本語だと投資対効果などと言ったりしますが、これはその投資に対してどれだけの利益が得られるのかを示す数値で、数値が大きいほど効果が高い投資だということを示します。計算式は下記の通りです。

$$ROI = \frac{R}{I}$$

簡単ですね。いくら Return が大きくても、Investment も大きいと、ROI すなわち投資効果は減ってしまうことがわかります。つまり、ビジネスにおいては単に Return があることだけが大事なわけではなく、Investment が極力少ないものである必要があることがわかると思います。

では、図 7.1において、Investment はどれで、Return はどれでしょう。Return というのはその活動の成果として得られる利益などを指すですから、この図で言えば”Return”は一番右の「アウトカム」だけとなります。わかりますか？つまりみなさんが作っているソフトウェアシステムは、作って納入するだけで価値があるわけではなく、運用されて、何らかの効果が得られて初めて「Return」、すなわち価値となる得るわけです。ソフトウェアシステムは、それ単体では「Investment」、つまり投資にすぎないということです。ええ～～そんなあ～。

そしてさらに、分母である Investment は小さい方が ROI は大きくなるので、みんなはなるべく仕事をしないほど良いという事になりますね。やったぜ！



▲図 7.2 ROI の最大化

でも別の見方をすると、アウトプットをいくらたくさん生み出しても、それがアウトカムにつながらなければ何の意味もないどころかお金の無駄遣いだったということになりますね。価値がゼロのものをどれだけたくさん作ってもゼロはゼロです。逆に言うと、大きなアウトカムが得られるならアウトプットは小さくシンプルであるほうが良いということになります。

7.2 スプリントの目的は「スプリントゴール」

前節からわかるとおり、スプリントの目的は価値を生み出すことであり、ソフトウェアを作ることではありません。スクラムでは、スプリントで生み出されるこの「価値」を、スプリントゴールとして設定するわけです。つまり、スプリントの目的は、より多くの機能を作るとか、より長時間働くとか、計画された機能をすべて完成させるとかではなく、価値を届けることだと言えます。スプリントゴールとはスプリントの「Why」です。『「なぜ」そのスプリントをやるのか？』です。つまり、スプリントゴールとは、そのスプリントが生み出そうとしている「価値」だということです。

ここで価値について考えてみます

筆者は大好きな LiSA(歌手)さんが Catch the moment のリリースイベントで手渡してくれたサイン入りポストカードを持っています(2枚!)。彼女はこのサインをものの数秒で書き上げるわけです。これなんて売れば数万円ぐらいの値が付いてもおかしくありません(絶対にそんなことはしませんが)。数秒で数万円、たとえば5秒で5万円とすると、1分で60万円(!)の価値を生むのです。ですがそんなことよりも、彼女とほんの短時間でも直接目を合わせて会話をしながら手渡してもらったという思い出が詰まっていて、実際のところ Priceless です。つまり何が言いたいかというと、価値というのは単純に作業時間や作業量とは単純にリニアに変換できるものではないということです。

前項の説明によると、スプリントゴールさえ達成出来るならば、スプリントプランニングで作ると決めた機能が、決めた通りにすべて完成していなくても良いことになりますよね？「目的が達成出来るなら手段はひとつじゃないよね？」ということです。

7.3 やらなくても良いことを最大化せよ

アジャイル宣言にはこういう記述があります。あえて英語版で。

「Simplicity--the art of maximizing the amount of work not done--is essential.」

筆者なりに意訳すると、「やらなくても良いことを最大化する技術が不可欠です」という意味です。やらないことの最大化、というのは、言い換えると、やることを最小限にしろということです。これは、上述の Investment を最小化せよという話と符合します。そのためにはシンプルさが必要だとアジャイル宣言は説いているわけです。一方で、いくらやることを減らすと言っても価値まで減らしてはいけないので、そのためによりシンプルなソリューションを考え、スプリントゴールで策定した価値を毀損することなく、やることを最小化すべしと言っているわけです。

たとえば、「当初フリーワード検索で考えていたけど、よく考えたら選択肢は5つしかないから、これだったらチェックボックスかラジオボタンで良くね？」となります。たぶんその方が文字コードチェックとかも要らないので実装も簡単ですし、なおかつ必要なものを検索するという当初の価値は全く失われていません。

これぐらいの単純な例なら良いですが、実際にはシンプルかつ価値の高いソフトウェアを作るというのはけっこう大変です。まず顧客が何を価値だと感じているのかをよく知ることが必要です。顧客は自分にとっての価値が何なのか、実はよく分かっていなかったりするので、単にヒアリングするだけではなく本当の意味で顧客に寄り添う必要があります。また、シンプルな設計とい

うのは実は最も難しかったりします。読みやすく理解しやすいコードだったり、結合度が低くて凝集度が高く、変更容易性の高い設計だったり、日々リファクタリングを重ねて常にシンプルさを保つ努力が必要だったりします。アジャイル宣言にもこう言及されています。

「技術的卓越性と優れた設計に対する不断の注意が機敏さを高めます。」

7.4 アジャイルでリカバリーってどうやるの？

冒頭のこの質問に戻ります。

極論すれば、アジャイルは「如何にして作らずに済ませるか」の技術です。よく「アジャイルでは何をやるよりも、何をやらないかの方が大事」といったことが言われます。やらなくて良いことはやらない、作らなくても良いものは作らない。いたずらに仕事の量を追うのではなく、その価値を見極めて本当に必要なことだけをやっていけば、物事はシンプルになる、ということです。そうしていかに少ないコードで大きな価値を出すか（つまり ROI）を考えていくと、作業時間も短くすることができるかも知れません。アジャイル開発では、そうやってリカバリーを行うのです。まあもちろんそれだけとは限らないし、それが上手く行かない場合もあるのは否定しませんが。

そもそも、冒頭の質問の背景には、従来型開発で骨の髄まで染みついた、「コストと納期とスコープは守り通さねばならない」という考え方があると思うのです。つまり、スプリントプランニングで作ると決めた機能は、スプリント終了までに決めた通りにすべて作らなくてはならないという考え方をしているのだと思うのです。

こういう考え方だと、遅延が出てしまった場合は残業をしてリカバリーするしかなくなってしまいます。残業は最悪です。残業は、なにか問題があったときに、安易にその場しのぎのリカバリーをする方法です。こういった方法が恒常化しているチームは、問題（この場合は遅延）の根本的な原因にリーチしようとしないので、改善していかないので。アジャイル宣言にある「サステナブル・ベース」にも反しますね。なので、生産性が上がらないと言っているチームのコーチに入ったときに、私が最初にコーチすることは、残業を禁止することだったりします。こうすることで、彼らはいかに残業をせずに問題を解決するかを考えるようになります。

7.5 まとめ

まとめます。

- 価値と量、価値と時間はリニアに変換できない
- ROI を高めるためには、仕事の量や時間は極力抑えよう
- シンプルな仕事をすることで、価値を毀損せずに量や時間を抑えることが可能となる
- リカバリーのために残業をするのは悪手



松永 広明 @qoragile <https://twitter.com/lsaconsul>
LSA CONSULTiNG 株式会社 代表取締役

<https://www.facebook.com/lsaconsul>

<https://www.linkedin.com/in/lsaconsulting/>

第8章

経験主義に基づく計画の立て方

松永広明

8.1 計画は何のためにするのか？

みなさんはどんなときに計画しますか？ 仕事の時はもちろんしますが、私生活においても、たとえば旅行に行くときなどに計画をしますよね？ ちょっと大きめのお金を使うときや、貯めるときなどにも計画することが多いと思います。

ではなぜ人は計画するんでしょうね？ ちょっと30秒ほど考えてみてください。

考えましたか？

答えは、「不確実性を（可能な限り）排除するため」です。たとえば旅行の計画では、確実に目的地に着いて、そこでさまざまなアクティビティを十分楽しむために計画を立てるわけですね。

これを念頭に下記を読んでみてください。

8.2 予実が乖離するのは計画が間違っていたから

一般的にアジャイルでは「予実管理」をしません。予実管理というのは、物事が計画（予定）通りに行ったかどうかを管理するという考え方ですが、アジャイルで予実管理をしないのは、物事が計画通りにいかない理由は、計画が間違っていたからに決まっていると考えるからです。

計画に実績を合わせるのは従来型の計画主義の考え方です。一方アジャイルは経験主義なので、過去の実績に合わせて計画を立てます。

みなさんはどちらが正しいと思いますか？ あるいはどちらが好きですか？

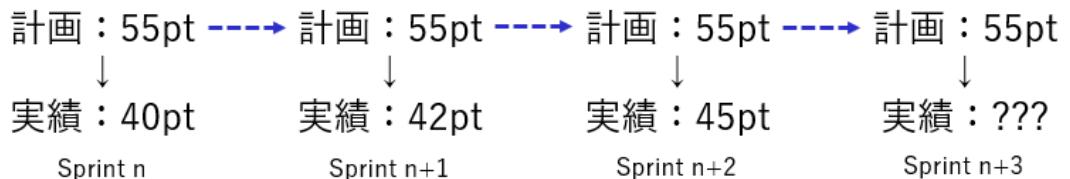
アジャイルが経験主義の立場を採る理由は、人間には未来を正確に予知する能力はない、ゆえに未来は不確実だ、と考えているからです。ましてや計画に実績を一致させようとする試みは、天気予報に合うように今日の天気を変えろと言っているようなもので、非合理としか言いようがないというわけです。なので、経験主義においては、実績が計画と乖離することは別に大した問題ではありません。乖離したなら、次からは計画の方を適応させれば良いからです。もう一度書きますが、予実が乖離するときは、計画の方が間違っていたのです。

8.3 計画主義による計画の立て方

たとえばスクラムにおいて、スプリントプランニングで今から取り組むスプリントバックログのスコープを決定する場合を例にとって、計画主義と経験主義の違いを図にしてみましょう。

たとえばスクラムでスプリントプランニングを行うとして、もし、計画主義の考え方でスプリントのスコープを計画するとしたら、下図のようになるでしょう。

計画主義では計画が正しいと考えるので、1スプリントで55ポイントやると決めたら、チームはその計画を達成できるように仕事を進めていくことになります。最初は計画通りに行かなくても、徐々に改善して計画を達成出来るようにするわけです。計画主義に慣れている方はこの考え方方に何の違和感も抱かないんじゃないかなと思います。



▲図 8.1 計画主義による計画の立て方

しかし、上記の例だとスプリントを重ねるごとに徐々に改善はしているものの、まだ一度も計画を達成した（予定通りに進んだ）ことがなく、このまま未達が続けばそろそろステークホルダーからのプレッシャーがかかってきそうです。チームはそのうちプレッシャーに負けて、見積もりそのものを不正に操作して（大きめに見積もる）、計画を達成したように見せかけるようになるかもしれません。

また、もしかしたらチームは、計画を達成するために残業することを選ぶかも知れません。が、残業して計画を達成しても、残業をすることになった原因が解消されたわけではないので、次のスプリントも残業することになり、チームは慢性的な残業体质に陥ってしまうでしょう。

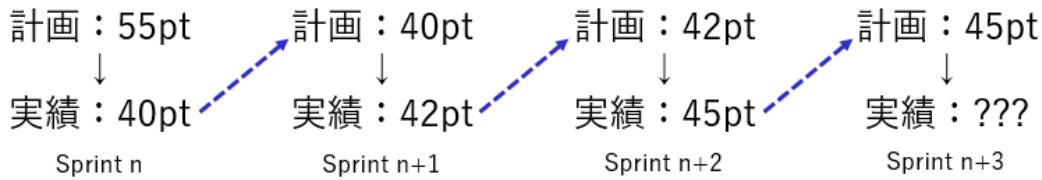
いずれの場合も、このやり方はチームの実態を表しておらず、スクラムで重要とされる「透明性」が崩壊する危機と言えます。

8.4 経験主義による計画の立て方

一方で経験主義では、過去の実績に基づいて計画を立てます。つまり、前回はこのやり方でうまくいったから、今回も同じやり方でだいたいうまくいくんだろう、と考えるわけです。

まず、Sprint n では、チームは 55 ポイントと計画しましたが、実際には 40 ポイントしか達成できませんでした。なので、次のスプリントでは、過去の実績に基づいて 40 ポイントと計画します。この時なにがしかの改善施策は入れておくようにします。そしてこのスプリントでは、改善施策が功を奏して、チームは計画を超える 42 ポイントを達成しました。なので次のスプリントでは 42 ポイントと計画します。これを繰り返して、常に実績に合わせて計画を立てるようしていきます^{*1}。

^{*1} 実際には過去 3 回のスプリントの実績を平均したものを次のスプリントの計画とすることが多いようです。



▲図 8.2 経験主義による計画の立て方

こちらのやり方では、毎回のように計画を超える実績を出せているため、チームのモチベーションはアガりっぱなしになります。ステークホルダーも大喜びです。

注目してほしいのは、このケースの場合計画主義でも経験主義でもでも出した実績は全く同じということです。にもかかわらず、チームのモチベーションには大きく差が出ます。モチベーションの違いは、そのうち大きな実績の違いとして現れてくるでしょう。

8.5 見積もりの精度が上がらない？

最後に、計画に不可欠な「見積もり」についてです。計画主義の考え方では、計画をより確かなものにするため、見積もりの精度を上げようとします。つまり見積もりと実績とを比較して、その差がなるべくゼロに近づくような取り組みをするわけです。が、これも実際にはあまり意味が無いのでやめたほうが良いでしょう。理由は上述の通り、人間には未来を正確に予知する能力は無いからです。

相対見積もりを採用している場合はさらに意味がありません。なぜなら、相対見積もりは大きさの比であり、例えば工数÷工数、あるいは時間÷時間などといった計算をすることになるので、見積もられた数値は無次元数（単位が無い）になるからです。単位のないものは測定できないため、予実の比較ができず、従って精度の計算もできません。「相対見積もりが難しい」「どうしても時間見積もりになってしまう」と言っているチームは、この罠にハマっていることが多いようです。

まとめます。

- 計画を立てることは、不確実性を（極力）排除することが目的
- アジャイル開発においては、未来は不確実と考えるため計画を絶対視しない
- 計画主義においては計画に実績を合わせようとするが、経験主義においては実績に基づいて計画を行う
- 未来は不確実なので、見積もりの精度を高めようとする取り組みはほとんど意味が無い



松永 広明 @qoragile <https://twitter.com/lساconsult>
LSA CONSULTING 株式会社 代表取締役

<https://www.facebook.com/lساconsult>

<https://www.linkedin.com/in/lساconsulting/>

第9章

アジャイルな案件を受託する組織職の一 事例

福田朋紀 (@chinmo)

僕はもう長いこと、受託開発の会社で、開発組織の組織職として働いていることになる。

うちの会社には営業がいないので、仕事を取ってくるのも僕の仕事だ。仕事を取ってきたら、部下に担当してもらう。

あらためて文章にしてみると、あまりにあっさりしていて、虚しさすら覚えるけれど、存外、僕はこの日々を楽しんでいる。その秘密は、「アジャイル」にあるんだ。

9.1 アジャイルという制約

アジャイルに関する仕事しかしない。

これはね、もう「そうすることに決めちゃった」としか言いようがない。色々理屈はつけられるのだけど、そういうた建前とか建て付けとは別にして、僕はもうそういう仕事しかしないし、周りにもきっとそうに違いないと思われている。周囲にそう考えてもらうまでには随分長い時間がかかったけど、一度そうなってしまえば、誰も僕に口を挟むことは無くなり、それどころかそいつの仕事は最初に僕に回してくれるようになった。

制約は、平凡な世界に刺激的な視点を提供し、一挙手一投足に、そうせざるを得ない理由を生む。結果として、僕という存在が、あるメッセージを周囲に受け取らせることになる。「あいつと関わると、好むと好まざるとに関わらず、アジャイルとやらに向き合わざるを得なくなる」というわけだ。

アジャイルな仕事がなかなかなくて、と嘯く組織職がいたとして、その人はつまるところ、「そうすること」にしていないだけなのだと思う。例えば僕の最初のアジャイルな仕事は、誰に求められたものでもなく、自分で提案したものだった。

アジャイルが向いていない、と誰かが考えている仕事があるとして、僕が「そうであること」をやめるかというとそんなことは一切ない。自分を曲げてまで何かに取り組むほど、勿体無い時間の使い方は無いじゃないか。それがソフトウェア開発であり、世の中に何かの価値を生み出すことを狙っているのなら、それは十分に複雑(Complex)で、アジャイルな自分はしっかりきっちりお役に立てるはずだ、というのが「そうすることにした」僕の揺るがないスタンスなのだ。はっはっは。

9.2 しかし部下はそうじゃない

アジャイルであることは自分で決めたことだけど、ただ、部下はそうじゃない。たまたま、すとんきょうな事を言っては周囲と波風を立てる奴の下につくことになっただけだ。ご愁傷様です。

別のところに行きたくなつたらいつでもどこに行っても良いのだよと言ってみたところで、僕たち日本人は相当な条件が揃わない限りブツブツモヤモヤグチグチしながらもそこに留まり陰鬱な雰囲気づくりに貢献しがちなわけだ。となると組織職としてできることは、

我々の「旅」が、そこそこ趣に満ちている事を目指すしかない。

お互い、楽しい！とか、やりがいがあるとか、そんな少年の様な爽やかなことが言える歳でもない。でもまあ、老後にポツポツと思い出しては人に話して聞かせたり、公園のベンチでフツ、とニヤついたりする様なことを体験してもらいたい。

9.3 苛烈な戦場で組織職ができるここと

ふと振り返った時に、何らかのポジティブな記憶を呼び起こす仕事を体験してもらいたい、というのが偽らざる想いではあるのだが、僕の部署のメンバーとして僕が取ってきた仕事を担当することは、極めて困難な仕事に挑むことを意味している。

例えば、仕様が決まっていない、要求が曖昧、日程感覚がズレている、顧客も課題も解決策もマーケットもフィットしていない、やりたいことに対して金が足りない、そもそもどうしたら悩みが解決するのかわからない、とにかく助けてほしい、この難局を乗り越えるための方法がアジャイルと聞いてここに来ました、などなど、靈媒師も頭を抱える様な危険な案件ばかりだからだ。

勝ち馬に乗るような仕事が無い、ということは、案件として担当するほぼ全てのプロダクト/サービスが、奮闘虚しく市場価値や問題の解決策を生み出せずに消滅する事を意味する。なんと、基本、負け戦前提なのだ！！

不安に苛まれ、それ聞いちゃダメだろ、みたいな問い合わせを投げかけ、利害関係者と衝突し、本気度合いのギャップに苛つき、形ばかりの作業に反対し、無茶苦茶なスケジュールに意を唱え、KPIの無い機能リリースロードマップを蹴り飛ばし、サポートへのクレームやネット上のバッシングに耐え、最後には人員削減や開発チームの解体を経て、サービス終了の屈辱に耐えながら少人数でのクロージング作業に従事する……そんな仕事したい奴なんているだろうか。

そしてそんな過酷な現場に放り込まれる部下たちに対して、他人のサラリーマン人生にそれなりに大きな影響を与える仕事を選択した組織職にできることとは、一体、何だろうか？

9.3.1 誰よりも楽しむ

目の前に広がるのが混沌とした戦場だとして、立ち尽くす部下たちの後ろから飛び出してきて「はいはい、盛り上がってますな！大好物です！」と真っ先に手を挙げて首を突っ込み、問題の構造を発見する度に小躍りし、何を面白いと思っているのか目をグルグルさせながらメンバーに話し、「絶対面白いから楽しんでおいで！」と送り出す、それが組織職の最初の仕事だと思っている。

面白そうだからやってみようぜ、と口説いている風には見えるが実際には上司部下の関係だ、任命しているに過ぎない。ただ、自分の熱意を相手にウザめに伝え、あわよくば相手の心に小さな火を灯したいのだ。

9.3.2 コンディションこそ全て

残業という概念を、自分の中から消そう。

部下の最高のパフォーマンスは、ポジティブな精神から生まれる。そのためには睡眠だ。部下は定時で家に帰っても、睡眠時間を削って何かしているかもしれないが、それは彼らの自由であり、組織職が彼らの睡眠時間を削って良い理由にはならない。

労働基準法ギリギリで部下の稼働を見事に回した経験があるなら、当たり前だが残業なしでも余裕で回せる。それだけで、部下のパフォーマンスは飛躍的に向上する。

ステークホルダーに残業してくれと言われても断ると良い。断れるか！と思うかもしれないが、普通に断れる。断ろう。当然、そのようなことを言われないように事前にさまざまな手を打つておくのがスマートだ。しかし覚えておいて欲しい。残業して稼ぐ一瞬の進捗は、リフレッシュした状態でやる場合よりも品質が低く、次のスプリントどころか翌日あっという間に取り返せる程度のものだ。

部下からもっと働きたい、とか言われることもある。そういう時は、その元気があるなら勉強して腕を上げて、同一時間で多くの成果を挙げられるようになって給与を上げるかもっと給与もらえるところに行けるようになれ、残業すると利益が下がるから君の評価も下がるぞ、と答えよう。

新人なら1年、ベテランでも3年くらいそうしていたら、定時になるとスッとチャットからいなくなってくれる。たまに残ってる奴は仲間ともうちょっとだけお喋りしたい勢だ。

9.3.3 仕事に学習を組み込む

契約の中に、この開発チームは業務を遂行するために必要な学習を行う、ということを織り込もう。と言っても当たり前だけど。仕事上でわからないことがあれば学ばなければならぬ。ソフトウェア開発は、サービス開発は、価値の創出は、事業の創造は、すべて日々の学びの繰り返しだ。

仕事中に学ぶ、まあそうだよね、と思うかもしれない。しかしあなたも一步踏み込んで、部下に、「就業時間中に積極的に勉強会やトレーニングをしよう、できれば毎日やろう。勉強会や研修や、カンファレンス参加や、社会貢献活動に費やす時間以外で、自分達の計画をし、ステークホルダーとコミットし、ソフトウェアを作ろう」と言うとしたらどうだろう。最初、部下は戸惑うはずだ。しかし、日々学び、学びを継続することを求めて学びの習慣をつけてもらうことが、彼らを自身に溢れるポジティブなエンジニア集団に変えていく。

目指すは、3-4時間集中してプログラミングしたら、契約分の成果を挙げるチームだ。あとは学びを深めたり、他者を助けたり、雑談しているのが理想だ。継続していくば、そのうち、少々要領が良いくらいのエンジニアでは追いつかないレベルの実力とチームワークがあるチームになると信じてやっている。

9.4 まとめ

アジャイルな案件を受託するなら、覚悟を決めて困難を楽しみ、部下の最高のパフォーマンスを引き出すことに集中し、責任を引き受けて学習に投資しよう。

大丈夫、誰も死がないし、責任は組織職が取ればよいのだから。



福田 朋紀 @chinmo <https://twitter.com/chinmo>

リコー IT ソリューションズ株式会社のアジャイル推し。アジャイル鳥取、JISA アジャイル開発 G、CoderDojo 鳥取チャンピオン、演劇企画夢 Ores、TRPG のキャリアが一番長くて 30 年以上

第 10 章

関係の質を改善させるためにやってよ かった 12 のこと

小糸 悠平 (@koito)

10.1 はじめに

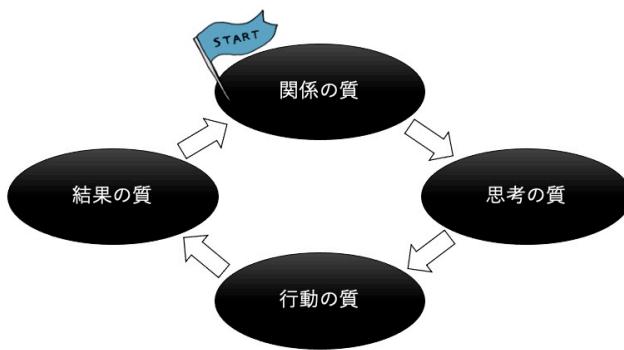
私は長年受託アジャイル（最近では共創アジャイルと呼ぶことも多いですが）のプロジェクトに携わってきました。共創アジャイルを実践してきた中で、アジャイル開発において組織・チーム・人との関係性がプロダクトの成功に極めて重要であるということに気付きました。

Daniel Kim 氏によって提唱された成功循環モデル^{*1}では、最初から成果（結果の質）を求めるうとするとチームが直接的な数字作りに走り、チーム内に摩擦や対立が生まれ（関係の質の悪化）、心理的安全性が確保されず（思考の質の悪化）、行動が消極的になり（行動の質の悪化）、さらに結果が悪くなる（結果の質の悪化）という失敗の循環^{*}が生まれると述べられています。

反対に、"成功の循環"として、チームの関係性が良くなることから着手する（関係の質の向上）と、考え方が前向きになることで多くのメンバーによる気付きが生まれ（思考の質の向上）、チームの積極的・自律的な行動に繋がり（行動の質の向上）、成果が生まれる（結果の質の向上）。成果が生まれることでますます関係性が良くなる（関係の質の更なる向上）と述べられています。

プロダクトを成功に導くために、上記のように多くの関係が必要となります。顧客と開発チームが密接に連携し、透明性とコミュニケーションを重視することで、より良い製品を迅速に提供することが可能となります。また、チーム内外の信頼関係と協力体制はプロダクトの成功に不可欠です。フィードバックを受け入れ改善を続けることで組織全体が成長し、より高い成果を生み出すことができるでしょう。

^{*1} 成功の循環 <https://www.humanvalue.co.jp/keywords/theory-of-success/>



▲図10.1 成功の循環

- ・顧客と一緒に本当に必要とするものについて考えられる関係
- ・開発者が顧客に必要なものを正しく理解できる関係
- ・開発したものに対し顧客にフィードバックをもらえる関係
- ・チームの活動をサポートし、モチベーションを高められる関係
- ・問題が発生した時にオープンに話し合い、迅速に解決できる関係
- ・チームメンバー同士が信頼し合い、協力して課題に取り組む関係、
- ・継続的な改善を追求し、成長し続ける関係

本章では、私が実際に取り組んで効果のあった関係性向上の手法を、チーム内およびチーム外の二つの視点から紹介します。全ての組織に適用できるわけではありませんので、各組織の状況に応じて判断していただければと思います。

10.2 チーム内の良い関係性の作り方

以下は、チーム内の関係性を改善するために実施した取り組みです。

やってよかったこと1：スクラムの勉強会を定期開催する

- ・内容：
 - チーム立上げ直後は、1スプリントに1時間くらいかけてスクラムについての勉強会を実施しました。
 - これまで実施してきたテーマは以下があります。
 - * 直近のスプリントで発生した事象に対するケーススタディ
 - * スクラム関連書籍の読書会
 - * スクラム関連イベントの動画鑑賞 などなど
 - * ワークショップの実施
- ・効果：
 - チームのスクラムへの理解度が一定の水準となり、議論の冒頭にチーム全員の認識を合わせる時間が減少します。これにより議論の本質に時間を割けるようになります。
 - メンバが知識を得ることでエンゲージメント向上にも繋がります。

やってよかったこと 2：競合システムの勉強会を開催する

- 内容：
 - チームが開発するプロダクトに競合製品がある場合は、その製品を皆で触って意見を交換する時間を設けました。皆で画面を操作し気付いたことを言い合い、自分たちのプロダクトの価値向上に繋がるものは PBI に追加していくというものです。
- 効果：
 - PO とチームが対等な立場で意見を交換できるので距離が縮まります。
 - 皆で意見を共有することでチームの一体感が強くなります。
 - PO がビジョンを共有しやすくなります。
 - サービスへの理解が深まり、チーム内の認識齟齬が減ります。

やってよかったこと 3：PBI をチームみんなで作る

- 内容：
 - 私たちのチームでは、立上げ時こそユーザストーリーマッピングを皆で検討したのですが、それ以降は何となく PO がユーザストーリーを考える人、開発者が開発する人というように、役割が決められてしまいました。その結果徐々に PO と開発者の関係が遠くなっていくのを感じたので、PO には PBI の完成度の低い（エピック単位までしか検討していない）状態でリファインメントに挑んでもらうことにしました。
 - リファインメントの中で、エピック単位のユーザストーリーマッピングを作成するワークを行いました。
- 効果：
 - チームでワークをすることで一体感が強くなります。
 - PO の負荷が下がり、開発メンバとのコミュニケーションの時間が確保出来るようになります。
 - PO がチームメンバを信頼して任せられるようになります。

やってよかったこと 4：(早く小さく) 失敗してみる

- 内容：
 - スプリントゴールを意図的に高く設定し、失敗を経験するようにします。4 スプリントに 1 回くらいの頻度で実施しました。
- 効果：
 - 失敗を経験することで、「失敗しても大丈夫」という心理的安全性が高まり、挑戦する雰囲気が生まれます。
 - ふりかえりのディスカッションが活発になり、改善のアイデアが多く生まれます。
 - ステークホルダーがチームの状況に興味を持ち、特にマネージャーはチームの状況をよく見るようになります。

やってよかったこと5：サービスを使ってもらっている所を見る

- 内容：
 - サービスイン後、利用者が使っているところを見学し、関係者とプロダクトの価値向上のためのワークを行いました。
- 効果：
 - ステークホルダとの距離が縮まります。
 - チームのプロダクトへの思い入れが強くなり、価値を高める議論が活性化されます。
 - 今後の開発に向けての意思統一が図れます。

やってよかったこと6：メンバが順番にイベントを欠席する

- 内容：
 - チームメンバが順番に休む。(休暇を取らない場合でもその日はスクラムチームに関わらない)
- 効果：
 - なんでも知っているメンバがいなくなるので、お互いに質問する環境ができる
 - 誰かに依存することなく、チームメンバ全員が自分の意見を発信するようになります。
 - 休んだメンバが翌日以降第三者視点でおかしな点に気付き指摘することができます。

やってよかったこと7：スプリントごとに小さな変化を入れる

- 内容：
 - 同じ状況が続くと、チームのメンバは飽きてきてしまうので、スプリントチームごとに変化を与えてみました。
 - これまでやってみた内容
 - * メンバの入れ替え(スクラムマスター⇨開発者の入替、複数チームであればチーム間でスクラムマスターを入替)
 - * スクラムマスターがスクラムイベントに参加しない
 - * スクラムイベントにゲストを招く
- 効果：
 - チームが変化に適応するためにチーム内の会話が増えます。
 - 変化に強いチームが出来ます。

10.3 チーム外(ステークホルダー)との良い関係性の作り方

プロダクトの成功には、チームを支える顧客やステークホルダーとの良好な関係が欠かせません。ステークホルダーとの関係性は、会社や組織の状況による差異が大きいため、抽象的な表現になりますがご理解ください。

やってよかったこと 1：ステークホルダーそれぞれの関心事を明確にする

- 内容：

- ステークホルダーと一口に言っても、開発部門長、事業部門長、企画、セールス、カスタマーサポートなど多岐にわたります。それぞれがプロダクト、チームに対して異なる期待や関心を持っています。
- 誰が何に興味を持っているのかを整理し、その関心内容に応じてステークホルダーをチームから適切な距離にいてもらうことが重要です。
- 図 10.2 のイメージのように一覧に整理しました。

役割	担当者名	業務上の責務	関心事(業務)	関心事(個人)
事業部長	Aさん	売上・利益の達成	・売上・利益を伸ばしたい ・売上・利益の予測を知りたい	・アジャイル開発の進め方
開発部長	Bさん	アジャイルの社内拡大	・アジャイル開発を詳しく知りたい ・アジャイルマインドを広めたい	・スクラムチーム内の雰囲気
企画	Cさん	新規施策検討	・顧客の需要を把握したい ・既存顧客の分析をしたい	・特になし
営業	Dさん	契約の成立、継続	・契約時の業務を簡素化したい ・顧客の傾向を把握したい	・顧客体験 ・今度の開発による自業務への影響
Ops	Eさん	定常業務の遂行	・定常業務を簡素化したい ・誰でも出来るようにしたい	・今度の開発による自業務への影響
カスタマーサポート	Fさん	顧客問合せ対応	・顧客満足度を向上したい ・顧客の問合せを減らしたい	・顧客体験 ・今度の開発による自業務への影響

▲図 10.2 ステークホルダーの関心整理

- 効果：

- ステークホルダーそれぞれの興味ある内容が明確になり、誰と何を調整すれば良いかがわかります。
- ステークホルダーとの効果的なコミュニケーションが可能になります。

やってよかったこと 9：ステークホルダーの必要十分なイベントを設定する

- 内容：

- ステークホルダーの関心事を整理した後、ステークホルダーの参加するイベント/参加しなくともよいイベントを決定しました@<fn>[comment]。
- 図 10.3 がイメージです。適切なスクラムイベントがない場合は、スクラムイベントに無理に参加してもらうわけではなく別の場を設けました。スプリントレビューの参加者はスプリントプランニングの時点で参加者を決定し、ステークホルダーに伝えることで早めに予定を確定させていました。

役割	担当者名	凡例：◎ 必須参加、○ Agendaにより必須(それ以外は参加不要)、- 参加不要			
		スプリントレビュー	意思決定Mtg	課題確認Mtg	情報共有Mtg
事業部長	Aさん	○	◎	◎	-
開発部長	Bさん	-	○	○	◎
企画	Cさん	○	-	○	-
営業	Dさん	○	-	○	-
Ops	Eさん	○	-	○	-
カスタマーサポート	Fさん	○	-	○	-
PO	Gさん	◎	◎	◎	◎
SM	Hさん	◎	○	○	◎
Dev	-	◎	-	-	-

▲図10.3 ステークホルダーの必要十分なイベントを設定する

- 効果：

- 各ステークホルダーにとって必要十分なイベントだけに参加するため、リソースが無駄になりません。
- ステークホルダーがイベントで求められることが明確なため、発言数が増加し、プロダクト・チームへの関心が高まります。
- イベント参加率が向上し、ステークホルダーが無理なくプロダクト・チームと関わることができます。

やってよかったこと10：スタートダッシュを成功させる

- 内容：

- ステークホルダーの中にはアジャイル開発に懐疑的な方やネガティブな感情を持つ方もいます。共創アジャイルの場合、私たちの会社の開発力にも懸念を持つ方もいます。この状態で序盤に開発が躊躇と、ネガティブな印象を持たれやすくなります。
- キックオフから最初の1ヶ月が重要で、そこでの成否がその後の関係性に大きな影響を与えます。結果を見せることが一番の方法です。
- 具体的には以下のようない行動をしていました。
 - * スプリントが始まる前のチームビルディング：ステークホルダーや開発者を雑談やイベント（ちょっとした外出など）に連れ出し、メンバ間の顔合わせや価値感を探ります。
 - * 1ヶ月で動くソフトウェアを見せることが難しい場合、チームビルディングより優先します。

- 効果：

- ステークホルダーがチーム、プロダクトに対してポジティブな感情になります。
- それにより、チームの活動に協力的になり以降の開発がし易くなります。

やってよかったこと11：スプリントレビューを好きになってもらう

- 内容：

- スプリントレビューはプロダクトの検査と適応を行う場です。スプリントレビューに

に対するステークホルダーの熱量を高めるために以下の工夫をしました。

- ワクワクするシナリオの検討
 - * 実際の利用イメージを持つるようにユースケースに沿ったシナリオを考えました。
- フィードバックがすぐに実装される経験
 - * POと相談し、ステークホルダーからのフィードバックを優先的に提供しました。
- 効果：
 - フィードバックがすぐに形になるので、ステークホルダーがレビューを好きになり、議論が活性化します。

やってよかったこと 12：イケてない状態を早期に曝け出す

- 内容：
 - 「やってよかったこと 10 スタートダッシュを成功させる」と反対の話になりますが、上手くいっていない状態も積極的に共有することでステークホルダーに味方になってもらいたい関係性を良くすることができます。
 - スプリントゴールの未達や、リリースバーンダウンの芳しくない状況、チームが抱えている課題などの透明性を高めるだけでなく、それに対して議論する場を作りチームと一緒に考えられる環境を作っていきましょう。
- 効果：
 - 早い段階での課題共有により、ステークホルダーと一緒に解決策を考えることができます。

10.4 まとめ

結果を出し続けられるチームを作るためには、チーム内/外、両方の関係の質を上げていく必要があります。

今回、12 のアイデアを紹介しましたが、皆さんのチームで実行している関係の質を向上させるためのアイデアについて情報交換させていただき、より良いチーム作りに繋げていければいいなと思っております。



小糸 悠平 (Yuhei Koito) https://twitter.com/koito_yuhei

KDDI アジャイル開発センター株式会社でアプリや Web サイト開発のスクラムマスターをやりながら、社内スクラムコミュニティを運営し組織内にスクラムを広げる活動中。

モットーは、”自分と自分の周りの人が幸せで居続けられるように行動する”。

第 11 章

不安とうまく付き合う

中村麻由

アジャイルに初めて取り組んだ時、期待していたような成果がすぐに出ず、焦りを感じたことはありませんか。プロジェクト序盤から問題が噴出し、むしろ進捗が遅く感じたことは。何より、周りから「アジャイルって速いんじゃなかったの？」などの批判を受け、チーム全体が意気消沈してしまったことはないでしょうか。

そして、なかなか改善する兆しが見られず、結局アジャイルの取り組み自体が終了してしまったことも……

プロジェクトの最初からつまづき、目に見える成果に辿り着けないと、チーム全体が不安に襲われると思います。しかしその不安が、実はアジャイルがうまくいっているサインだと考えるとどうでしょうか。

職場で初めてアジャイルが導入された際、私も同僚もかなり苦戦をしました。うまくいっていないかった原因は様々でしたが、当時、クライアントを含めチーム全体を覆いがちだったネガティブな雰囲気をよく覚えています。それは「不安」、そしてそれが解消されないことによる「不満」でした。

アジャイルを学ぶための書籍や研修では、主に「何をするべきか」を知る事ができます。しかし、チームとしてアジャイルのマインドセットを育み成長する過程で、誰もが体験しがちな「感情」についてはあまり触れられません。講習を受け、体系的な知識を得たチームですら実戦の中で不安を感じるならば、アジャイルのことをあまり知らずにステークホルダー（利害関係者）になった人は、更に大きな不安を感じるかもしれません。

アジャイル文化を根付かせる鍵は人間同士のコミュニケーションや信頼関係です。そのため、誰かが感じている「不安」など、感情に向き合う事は大切な一歩となります。筆者は主に実践を通してアジャイルを学びましたが、本章ではデザイナーという立場から見えた景色と学び、そして初めてアジャイルに取り組むチームが陥りがちな「不安」について考察します。

11.1 とあるチームのストーリー

アジャイルを導入したばかりのプロダクトチームが、スプリントデモを始める場面を思い浮かべてください。今回はスプリント 2 が終わったところです。

PO が緊張した面持ちで、デモを始めようとしています。スプリント 1 のデモは特に見せられるものがなく、気まずい雰囲気で終わってしまいました。デザイナーと開発者も参加していますが、

なにか揉めているように小声で会話を交わしています。部屋にステークホルダーが入ってきました。デモが始まります。

まずは今回のスプリントで取り組んだユーザーストーリーのデモです。アプリを立ち上げるといいくつかのカード型 UI が表示されました。カードの中に表示されるメッセージはまだダミーのテキストで、見た目も簡素です。「これがデザインなの？」ビジネス部門から参加していたステークホルダーの一人がコメントをしました。「あ、いえ、まだデザインは反映されていません。」慌てた様子でデザイナーが答えます。「デザインが仕上がってこないので、後回しにして機能だけ先に作っています。」開発者が続けました。ステークホルダーは少し不満そうな表情を見せています。「デモは以上です。デザインの方から、次のストーリーのモックアップを見せられますか？」PO に聞かれ、デザイナーは少し焦った感じで返事をしました。「あ、はい…。」静まり返った時間が流れます。デザイナーが、しどろもどろになりながらデザインプロトタイプを見せ始めます。「まだラフなスケッチですが…どうでしょう？」「ボタンの位置をあと 2 ピクセル、上にあげた方がいいんじゃない？」デザインマネージャーがコメントします。「あ、でもまだラフなので…。」結局、特に有意義なフィードバックもなく、予定していた時間より 20 分早く会は終了しました。あまり盛り上がることもなく、次第に次のデモから参加する人は減っていきます——

11.2 不安全感に負けてしまうチームと周りの人たち

これは、私が関わった複数のプロジェクトで体験したデモの様子を組み合わせたものです。デザイナーは、デザインに必要な時間が十分与えられていないと感じ、まだ自分の中でも納得がいっていないものを見せ、それに対して否定的な意見がくることに不満でした。開発者はデザインに時間がかかりすぎていると感じ、開発を進められないまま時間が過ぎていくことに苛立ちを感じていました。PO は、スプリント内で終わると踏んでいたストーリーが終わらず、焦りを感じ、デザインと開発者のすれ違いに気付いていませんでした。ステークホルダーは、アジャイルを「とにかくスピードがあがり、ものすごいプロダクトがパッと出てくるもの」という印象を持っていました。

何より、「アジャイルは正しいはず」という空気に、誰も「うまくいっていない」という声をあげられませんでした。結果思ったように開発は進まず、次第に自信を無くしたチームは、アジャイルはもうやめたほうがいいんじゃないかと思い始めました。

しかし、本当はこのチームはいいスタートを切っていたはずです。

11.3 リスクを早く炙り出した方がいいアジャイル

「アジャイルな見積りと計画づくり～価値あるソフトウェアを育てる概念と技法～」という本のなかで Mike Cohn 氏は、従来の計画方法が崩れがちな理由の一つに「不確実性を無視している」ことをあげています。

「まず、私たちはプロダクトに関する不確実性を無視している。初期の要求分析において、完璧で漏れのないプロダクト仕様を策定できることを前提にしている。」

「同様に、私たちはプロダクトを構築する方法に関する不確実性も無視している。どんな作業なのかはっきりわかっていないのに、たとえば「10 日間」などと、高い精度で見積もれるという前提に立っているのだ。」

プロジェクトの初期段階におけるプロダクトの不確実性はみなさん意識することが多いと思います。ですが、「プロダクトを構築する方法に関する不確実性」は、むしろ見て見ぬふりをしたくなる部分かもしれません。どのくらいかかるか、どうアプローチするのか、まだ正確にはわかりませんと口に出してしまうと、まるでチームの知識や専門性が足りていないように感じるからです。ですが、プロダクトに不確実性があるということは、それを構築する方法にも少なからず不確実性があるということです。

また、プロダクト開発がチームスポーツであること、私たちが人間であるということも、不確実性を上げる要素です。特に新しくできたばかりのチームでは、お互いのペースを掴むまでの試行錯誤も不確実性の要素になるでしょう。

逆に、アジャイルな計画づくりがうまくいくためには、この不確実性を受け入れることが大切な第一歩となります。前出の「アジャイルな見積りと計画づくり～価値あるソフトウェアを育てる概念と技法～」では、最後の章で架空のチームが登場し、様々な課題に直面しながら計画を修正していく様子が描かれています。その様子を見ていると、チームがプロダクトについてだけでなく、自分たちについても少しづつ学び、チームワークの質を上げていく様子を見ることができます。

このチームのように、アジャイルでは早い段階で「プロダクト」と「プロダクト構築の方法」、両方に関して「うまくいっていないこと」を可視化させ、対処することでリスクを減らしていくことが大切です。しかし、実際には不確実性が多い中で問題が噴出すると、人は不安になるものです。

11.4 「デザイン」が更に不安を募らせる

デザインと開発の協業経験の有無によっても不安度は変わります。今まで「デザイン仕様」を受け取ってから開発を始めていた場合、真っ白な状態からデザインが始まることに不安を感じる人は多いでしょう。

もちろん、リリース用の開発が始まる前に「ディスカバリー」を行い、ある程度コンセプト作りをしているケースがあるかもしれません。しかし、おおまかな方向性が定まっていたとしても、デザインという作業は発散と収束を繰り返します。しかも一度ではなく、視点を変えながら何度もその過程を繰り返すことで最終的なソリューションに落ち着きます。つまり、発散させている間は常に「何が正しいのかわからない」という状態が続くのです。

開発も模索、検討する時期があるとは思いますが、最短で最適解を目指すのではないでしょうか。相対的な差ではありますが、お互い、その時々のベクトルが違うことが意識できていないと、次第に不満が出てきます。デザイナーは十分な時間がないと感じたまま開発者を待たせているプレッシャーに悩み、開発はなかなかソリューションが決まらないことに苛立ちを覚えます。

11.5 不安全感とうまく付き合うには

こう聞くと「ならばデザインと開発は別に作業したほうがいいのでは?」と思うかもしれません。しかし、その場合「技術的に実現できないデザインをしてしまう」「開発段階でデザインが改変され、元の重要な意図が抜けてしまう」などのリスクを後回しにすることになります。もちろん、プロジェクトに余裕があれば「出戻り」が起こり、デザイナーが修正することになりますが、最悪の場合、重要な要素が抜けたことに気づかずそのままリリースされることになります。一見非効率に見えるデザインと開発が一体になったアジャイルは、長い目でみれば効率とプロダクトの質を両立する最適解です。

鍵となるのは、いかに素早くプロダクトチームとして成長し、お互いのリズムを掴むか。初めのうちに疑問点や不協和音を見ないフリをしてしまうと、あとあと自分たちに戻ってくることになります。つまり不安になることが起こった場合、多少時間がかったとしても未来の自分たちを助けるつもりで対処する方がいいのです。

それでは、不安とうまく付き合いつつ、前に進むにはどうすればいいのでしょうか？全ての不安を取り除くのは難しいかもしれません、筆者が体験した過去の取り組みでうまくいった例をいくつかあげてみます。

11.5.1 フレームワーク通りにやっているのに、しっくりこない時

アジャイル初心者のチームにありがちなのが、研修で習ったプロセスをそのまま再現しようとし、理想通りに進めず「失敗した」と思ってしまうことです。あくまで理論は理論であり、実践では予測通りに行かないのは当たり前のことです。その際、チームが不安を口に出せずプロセスに従うことが強制されると、息苦しい時間が続くことになります。

振り返りで「うまくいっていない」と口に出せる、心理的安全性が大切です。また、上がってきた声を「でもアジャイルってこういうものだから」と潰してしまわない、思考停止しないことも重要でしょう。アジャイルの原則から離れすぎてはいけませんが、習ったプロセスに固執するのではなく、自分たちに最適な方法を考えましょう。どこまで崩していいのかわからないこともあるでしょうから、アジャイルコーチなどに参加してもらうのも有効です。

11.5.2 デザインがボトルネックに感じる場合

デザインがボトルネックになる状況を防ぐために、いくつかできることがあります。

- ディスカバリープロジェクトから定期的に開発者が参加し、コンセプトのイメージを掴んだり、技術的な見解をインプットする
- 体験設計のビジョンになるもの（ストーリーボードなど）を誰でも、どこからでも見られるようにしておく
- ムードボード、UIのサンプルなどをラフに作っておき、何となく出来上がりの雰囲気を掴んでおく
- デザインシステムやガイドラインを準備しておき、ユーザーストーリーの取り組み始めから参照できるようにしておく
- 少少ドキュメンテーションが後回しになってしまっても、コミュニケーションを取りお互いの理解を深めることを優先し、ドキュメンテーションは合意したことの記録程度にとどめる

密なコミュニケーションも重要です。デザイナーの中で決まっていない要素はどこなのか、それはなぜか。開発者は先に作業できること、後回しにしてもいいことなどを伝えると、デザイナーの作業も進みやすくなります。

11.5.3 チームの周りに不安感が漂う場合

チーム外のステークホルダーとの調整は難しいかも知れませんが、できることはあります。

- 「アジャイルとは？」「デザインとは？」といった勉強会を、社内ステークホルダー向けに開

催する

- デモの中でデザインモックアップを見せる時間をきちんと組み込む
- ある程度形になってからデモにステークホルダーを呼ぶ

ここでも鍵となるのはコミュニケーションです。自分たちが何を意図していて、どんなものを目指しているのか、なるべく具体的に見せられる何かを用意をしましょう。

11.6 終わりに

プロダクトチームのメンバー、また周りにいるステークホルダーも一人ひとり人間です。不安になることもあるし、アジャイルの本が示す型にはまりきらない部分も多いでしょう。口に出せない不安感、さらに不信感が積み重なると「プロダクトを構築する方法に関する不確実性」が増すことになります。

「小さく始め、改善させながら積み重ねていく」ことは、チーム自体の成長にも言えることです。まだ取り組み始めたばかりのチームが、フレームワークを試してみてもうまくいかなくて不安に思うことは当たり前のことです。誰かが不安を感じていたら、それはチームがステップアップするチャンスです。その理由に注目して、どうしたら解消できそうか考えてみてください。

不安が「チャンス」に見え始めたとき、みなさんはアジャイルな組織として大きな一歩を踏み出した証拠です。

参考書籍

MIKE COHN (2009). アジャイルな見積もりと計画づくり 毎日コミュニケーションズ
Collin Lyons (2019). Make Learn Change ustwo ltd.



中村麻由@mayunak <https://twitter.com/mayunak/>
<https://mayunak.com/>

ustwo Tokyoでプリンシパルデザイナーをしています。イギリスに10年滞在し、ユーザー中心設計を学んだあと現在の会社のロンドン本社に入社しました。デザインと開発の距離をもっと縮めたいと考えています。

第 12 章

アジャイルは反復してナンボ！

松永 広明

12.1 アジャイルとはなんぞや？

まずアジャイルとはなんぞや？ というところから考察してみたいと思います。
アジャイル宣言をひもとくと、こんなことが書いてあります。

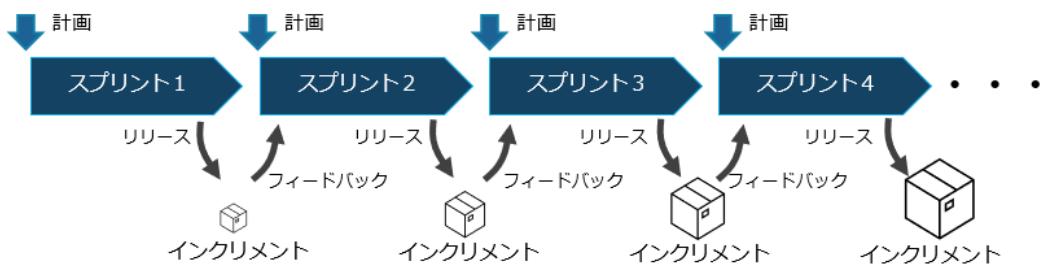
「計画に従うことよりも変化への対応を」

また、同じくアジャイル宣言 12 の原則には下記のような記述もあります。

「顧客満足を最優先し、価値のあるソフトウェアを早く継続的に提供します」

「動くソフトウェアを、2-3 週間から 2-3 ヶ月というできるだけ短い時間間隔でリリースします」

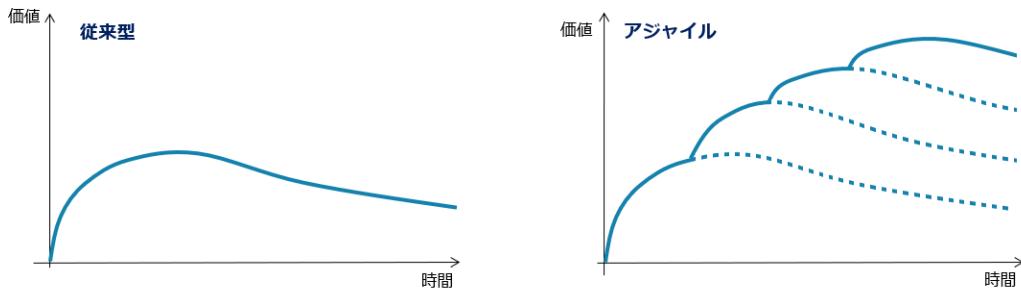
今や 2-3 ヶ月を「短い時間間隔」と言ってしまうのは議論ありそうですが、そこは 20 年も前の話ですので置いておくとして、つまりここに書かれていることは、図に示すと下記のようなイメージです。



▲図 12.1 アジャイルの代表的なプロセス

言葉で言い換えるなら、価値を見定めつつ、定期的にインクリメントをリリースしてフィードバックを獲得し、それを反映してさらなる価値を付加したインクリメントをリリースし、これを繰り返す。つまり反復することでプロダクトの価値を高め続けること（図2）、これがアジャイルの

価値であり、原則であると言っているわけです。



▲図12.2 従来開発型とアジャイル開発における価値の推移

一方で、例えばウォーターフォールのような従来型の進め方では、開発が進むにつれて価値が高まっていきますが、その間にも新しいテクノロジーの発明や目新しいUXの創造などにより市場はどんどん変化していくため、プロダクトの価値は徐々に劣化していくことになります（図12.2左）。

12.2 「反復」とは何か

人間は「反復」によって多くの技能を獲得します。スポーツでも習字でも音楽でもそうですが、反復練習なくして上手になることはありません。そして、反復周期は短いほど、つまり頻度が高いほど一般に練習の効果は高いです。例えば、年に1回ピアノを弾いたってなかなか上達はしません。しかし週に1回、できれば毎日練習すると、上達はずいぶん早くなることは、想像に難くありません。

そして、アジャイル開発においては、反復がうまく行っていることを示す最も重要な指標は、毎回のイテレーション（スプリント）でチームがなんらかの価値あるインクリメントを完成させていくことです。ソフトウェア開発においては、価値あるインクリメントとはデモができるインクリメントのことです。逆に、顧客への実動作デモができないなら、それは顧客価値があるとは言えません。なぜなら、実際に動作するところを確認できないと、顧客はそのソフトウェアに価値があるかどうかの判断ができないからです。また、価値判断が出来ないということは、フィードバックを返すことができないということでもあり、つまりここでフィードバックループは途切れてしまい、図12.1や図12.2のような、アジャイルの重要な特徴の実現が難しくなってしまうのです。

逆に言うと、イテレーションを何回も繰り返さないとデモ可能なアウトプットを出せないチームは、反復がうまく行っていない可能性が高く、これはすなわちアジャイル開発がうまくいっていない可能性が非常に高いと言えます。

12.3 「反復」なきアジャイル

ところが、最近よく見る自称「アジャイル」は図12.3のようなものが多いです。

一応イテレーション（スプリント）は一定周期で区切っているし、スクラムイベントも定期的にやってる（たぶん）のですが、聰明なこの本の読者ならすぐにお気づきなのではと思いますが、こ

これはアジャイルではないですよね。はい、おっしゃる通りです。これは「アジャイルに見える」だけの単なるウォーターフォールですね。プロダクトのリリースは最後の1回だけですし、フィードバックなんてどこにも取り込んでないし、価値を高めてもいない。でも残念ながらこういう似非アジャイルが巷にあふれているのは事実です。

この方法では反復はまったく機能していません。例えばゴルフのスイングを練習するには、自分のスイングを動画に撮って、1回～数回振るごとにビデオをチェックし、良くないところを直してまたクラブを振る、というフィードバックループを繰り返すことが効果的なのですが、図12.3の方法では、そのループが回っていません。つまり、図12.3のような状態は、何かを習得するには非常に不利な状態と言えます。



▲図12.3 残念なアジャイル（アジャイルではない）

この状態に陥っているチームは、プロダクト開発そのもの以外もうまく行きません。なぜなら、上述したとおり人間は反復によって技能を習得するからです。例えば、心理的安全性にしても、自己組織化にしても、T字型スキルにしても、「反復（イテレーション）」をする中で徐々に習得されていくものだからです。

つまり、このままの状態では、スプリントの回数を重ねても反復効果が出づらく、したがってチームやプロセスやプロダクトに改善を期待することが難しくなります。改善のためには根本原因にリーチする必要があります。

ではなぜこうなるのか？ 原因はプロダクトバックログづくりにあります。

12.4 プロダクトバックログは「プロダクト」のバックログ

プロダクトバックログは、その名の通り「プロダクト」の「バックログ」です。何が言いたいかというと、プロダクトバックログアイテム(PBI)は、プロダクトを分解したものである必要があるということです。プロダクトバックログアイテムは、そのひとつずつが何らかの「価値」を顧客やユーザーに提供できなくてはなりません。「価値」というのは、例えばユーザーが意識的あるいは無意識的に抱えている何らかの問題を解決するとか、新しいことができるようになるとか、今まで以上にお金が儲かるようになるとか、そういうことです。

この分割方法は、よくケーキにも例えられます。よく知られたメタファなので、ここでは解説しません。ご存知ない方は調べてみてください。すぐにたくさんの記事が見つかると思います。

そうして分割されたプロダクトバックログが実際に作られ、リリースされることで顧客やユーザーは実際にそれを使ってみることができるようになり、使ってみて初めてフィードバックを返すことができるようになるわけです。図12.3のようなやり方だと、開発序盤から中盤までは、例えば要件定義書や設計書を見せるのが関の山で、これでは顧客からのフィードバックを得るのはほとんど不可能です。そりゃそうですよね、例えばカメラの要件定義書を見て、「このカメラ使いやすいですか？」「画質はどうですか？」などと聞かれたって答えようがありません。

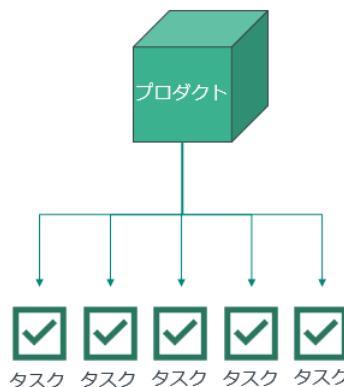
12.5 プロダクトバックログは ToDo リストではない

上記のような似非アジャイルになっているチームのバックログをみてみると、バックログアイテムは、たとえば下記のようになっていたりします。

- ・ ○○システムのデータフローをレビューする
- ・ 撮影画像確認画面の設計をする
- ・ ユーザープロファイル確認画面についてデザイン部と合意する

どうでしょうか？ これらはそれ単体でユーザーに価値をもたらすでしょうか。「データフローのレビュー」はユーザーにどんな価値がありますか？ 画面デザインへの合意は、顧客のどんな問題を解決するでしょうか。これらは、それ単体では価値とはならず、その他の多くの作業がつながって初めて何かの価値を生むようなつくりになっています。

これを図に示すと上図のようになります。つまり、プロダクトをいきなりタスクに分解しているわけです。上述の通り、タスクはそれ単体では価値とはならないため、顧客にリリースすることはできません。これは単なる ToDo リストであり、プロダクトバックログではありません。このような作り方をすると、すべてのタスクが完了するまでプロダクトはリリース出来ません。こういう開発のやり方をなんと呼ぶかというと、そうです、ウォーターフォールと呼ぶわけですね。



▲図 12.4 プロダクトをタスクに分解している

Product backlog ≠ To Do list

▲図 12.5 プロダクトバックログは ToDo リストではない

12.6 小さな価値とは

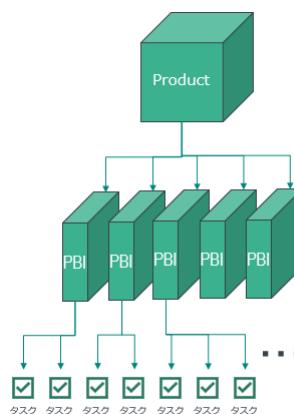
一方で下記のようなバックログアイテムはどうでしょう？

- 通知ボタンを押下すると「通知ON」ランプが点灯する。
- チャットボタンを押下するとチャットウインドウが開く。

これらは、機能としてはとても小さなものに過ぎませんが、それ単体で十分な価値があります。通知がONになっているかOFFになっているかは、ユーザーにとってそれなりに重要な情報です。プロダクトバックログアイテム(PBI)は単体ではこのような小さな価値が提供できれば十分なのです。

これを図にすると図12.6のようになります。

この図ではプロダクトを一旦小さなプロダクト(=価値のあるもの)に分解しています。それは小さいが故にシンプルで、すなわち早く完成でき、早くリリースでき、その価値はすぐにフィードバックを得ることができます。必要があれば、これらの小さなプロダクト(バックログアイテム)をさらに小さなタスク(=作業)に分解しても良いでしょう。



▲図12.6 プロダクトをより小さなプロダクトに分割している

つまり、こういった形でプロダクトバックログをうまく作ることができれば、反復はうまく回るようになるのがわかると思います。反復がうまく回り始めれば、本線のプロダクト開発だけでなく、チームの改善もうまくいくようになり、さらにその効果でプロダクト開発がよりうまくいくというループが回るようになります。

実は、プロダクトバックログづくりは初めてアジャイルに取り組むチームが最初にぶつかる壁のひとつです。図12.4に示したアンチパターン以外にも、プロダクトバックログアイテムのサイズが大きすぎて、完成までに1ヶ月～数ヶ月かかるというのもよく見かけます。インクリメントが付加する価値は、上述したように「ボタンを押すとランプが点く」ぐらいの、本当に小さなもので良いのです。試してみてください。

12.7 まとめ

- 人間は反復によって技能を獲得する
- 反復がうまく回らないと開発そのものもプロセスの改善もまくいかない
- 反復するためには、プロダクトバックログアイテムは、「プロダクト」を分割したものである必要がある

- プロダクトバックログアイテムには ToDo リストではない
- プロダクトバックログアイテムは本当に小さな価値で良い
- アジャイル開発において、反復がうまく行っている最も重要な指標は、毎回のイテレーション（スプリント）でデモができることがある

上手にプロダクトバックログを作つて、反復を回していきましょう！



松永 広明 @qoragile <https://twitter.com/lsaconsul>
LSA CONSULTiNG 株式会社 代表取締役

<https://www.facebook.com/lsaconsul>

<https://www.linkedin.com/in/lsaconsulting/>

第 13 章

デザインから逆算して難易度を見積もるための観点

酒井文也 (@fumiyasac)

13.1 はじめに

私はこれまで、iOS と Android の両方でモバイルアプリ開発を経験してきました。そのなかで、プロジェクトの成功において、デザインを基に実装の難易度を見積もることがいかに重要であるかを強く実感しています。iOS と Android は、それぞれ異なるデザインガイドラインや技術的な要件を持っており、同じ（ように見える）デザインを実装する場合でも、プラットフォームごとに異なるアプローチが必要です。

本記事では、デザインから逆算して難易度を見積もる際に考慮すべき重要な観点について、私の経験を踏まえて解説します。これらの観点を押さえることで、開発チームがデザインと技術的実現性のバランスを取りながら、効率的かつ高品質なアプリを開発するためのヒントを提供できればと思います。

13.2 モバイルアプリ開発において難易度を見積もる際に重要な観点を整理する

モバイルアプリ開発において、デザインから逆算して実装の難易度を正確に見積もることは、プロジェクトの円滑な進行に不可欠です。以下の観点を考慮することで、より精度の高い見積もりを実現できます（※プロジェクトの内容や事情によっては、本稿で列挙する観点以外でも重要なポイントとなり得る点はあると思います）。

1. UI コンポーネントの複雑さ

UI コンポーネントの選択は、開発の難易度に大きな影響を与えます。標準の UI コンポーネントで実現できる場合は、開発の手間が軽減されますが、カスタム UI コンポーネントが必要な場合は、実装とテストに時間がかかります。また、アニメーションやレイアウトの複雑さもプロジェクトの進行を左右します。シンプルな遷移や配置よりも、複雑なインタラクションや階層構造を持つデザインの方が当然必要な工数は多くかかる事になります。

2. プラットフォーム間の差異

iOSとAndroidでは、UIガイドラインやプラットフォーム固有の機能に違いがあります。これらの差異を正しく理解し、両プラットフォームでの一貫性を保ちながら、デザインを適切に実装することが求められます。また、デバイスごとの画面サイズやアスペクト比の違いにも対応する必要があります。これにより、プラットフォームごとのユーザー体験を最適化し、開発の手戻りを減らすことができます。

3. データフローとステート管理

アプリケーション内のデータフローと状態管理の複雑さも、実装難易度に大きく影響します。単純な画面遷移や状態管理であれば、開発はスムーズに進みますが、複数の依存関係が絡む場合は、慎重な設計とテストが必要です。データの受け渡しや共有の方法を事前に明確にすることで、後のトラブルを回避できます。

4. パフォーマンスへの影響

アプリのパフォーマンスはユーザー体験を左右するため、リストビューでの大量データの扱いや、画像処理、動画再生などの重い処理を効率的に実装することが求められます。また、バックグラウンド処理やプッシュ通知の実装も、アプリの応答性やバッテリー消費に影響を与えるため、慎重に設計する必要があります。

5. API連携とデータ同期

API連携の複雑さや、オフライン対応の必要性は、開発の難易度に直結します。必要なAPI呼び出しの数やデータの同期方法を事前に計画することで、ネットワークの不安定さやデータの競合を防ぎ、スムーズなデータ同期を実現します。

6. セキュリティ要件

ユーザー認証やデータの暗号化、セキュアなストレージの実装は、アプリのセキュリティを確保するために不可欠です。これらの要件は、実装の複雑さに影響を与えるため、開発の初期段階から考慮する必要があります。

7. アクセシビリティ対応

アプリが多くの中年層にとって使いやすいものであるためには、アクセシビリティ対応が重要です。色のコントラスト比や音声読み上げ対応、タッチターゲットのサイズなど、アクセシビリティに関する要件を満たすことで、アプリの品質とユーザー満足度を向上させることができます。

8. ローカライゼーション

多言語対応や右から左への表記（RTL）の対応、地域ごとの法的要件や文化的配慮は、グローバル展開を視野に入れたアプリ開発において重要です。これらを適切に考慮することで、さまざまな市場での成功を促進します。

9. テスト容易性

テストの容易性は、プロジェクトの進行に大きな影響を与えます。ユニットテストや UI 自動テストの実装しやすさ、エッジケースや異常系のテストシナリオを早期に設計することで、後のバグ発見や修正を最小限に抑えることができます。

10. メンテナンス性と拡張性

コードの再利用性や、将来的な機能追加や変更のしやすさを考慮することは、長期的なプロジェクトの成功に不可欠です。適切なドキュメンテーションを行い、メンテナンス性と拡張性を確保することで、プロジェクトの持続可能性を高めます。

これらの観点を考慮することで、デザインから逆算して実装の難易度をより正確に見積もることができます。プロジェクトの円滑な進行につながります。

13.3 iOS・Android それぞれの UI 実装観点からの重要性

私自身はモバイルアプリ開発の中でも、特に UI 実装に関連する分野が特に関心が高いです。この点については、20代の時にデザイナーとしてのキャリアを歩んだ事も関係していると思います。故にモバイルアプリ開発において、UI 実装の観点から見る難易度の見積もりは、プロジェクトの成功に直結する重要な要素であると考えています。

1. コンポーネント単位で見た時の方針選択

UI 実装において、標準コンポーネントで済むか、カスタム開発が必要かで、工数やリソースの配分が大きく異なります。特に、カスタム UI コンポーネントの実装は、描画処理やタッチイベント処理の複雑さがアプリのパフォーマンスとユーザー体験に直接影響を与えるため、慎重な計画と見積もりが求められます。また、プラットフォーム固有の API を適切に活用することで、効率的かつ効果的な実装が可能となります。

2. iOS/Android 間における考え方の相違点

iOS と Android では、UI ガイドラインやレイアウトエンジン、ナビゲーションパターンなどに大きな違いがあります。これらの差異を理解し、それぞれのプラットフォームに最適化することが、スムーズな実装と高いユーザー満足度を実現する鍵となります。例えば、iOS と Android では画面遷移の基本的な考え方が異なるため、適切なナビゲーション設計が必要です。

3. カスタム UI コンポーネントの実装

カスタム UI コンポーネントの実装においては、プラットフォームごとの最適化が重要です。描画処理の最適化やタッチイベント処理の複雑さを考慮することで、アプリのパフォーマンスを向上させ、ユーザー体験を高めることができます。また、プラットフォーム固有の API を理解し、活用することが、高品質な UI を実現するための重要なステップです。

4. アニメーションとトランジション

複雑なアニメーションやトランジションの実装は、UIの滑らかさとインタラクティブ性を高めるために不可欠です。しかし、その実装には、プラットフォーム固有の最適化テクニックが必要となります。特に、アニメーション設計においては、パフォーマンスを考慮した工夫がユーザー体験の向上に直結します。

5. システム UIとの統合

システム UIとの統合、例えばステータスバーやナビゲーションバーの扱いは、アプリ全体の見た目と操作性に大きな影響を与えます。iOSのエッジスワイプジェスチャーなど、プラットフォーム固有の動作への対応も、実装上の重要なポイントです。

6. アプリのライフサイクルの違い

iOSとAndroidでは、アプリのライフサイクル、特にバックグラウンド処理やメモリ管理において違いがあります。これらの違いを理解し、適切に対応することで、アプリの安定性と性能を向上させることができます。特に、複雑なアプリケーションでは、これらの要素を無視すると、パフォーマンスの低下やクラッシュの原因となる可能性があります。

これらの観点を踏まえた難易度見積もりは、プロジェクトの計画とリスク管理を大幅に改善します。また、デザイナーとの協業においても、これらの技術的な制約や可能性を共有することで、実装しやすく、かつユーザー体験の高いデザインの創出につながります。結果として、アプリの品質向上と開発効率の最大化が達成され、プロジェクト全体の成功に寄与します。

13.4 アジャイル開発の観点から iOS/Android での類似点・相違点を押さえる例

iOS/Android両方のUIガイドライン、レイアウトエンジン、ナビゲーションパターンなどの違いを事前に把握しておくことで、各プラットフォームに応じた開発タスクの見積もりが正確に行う事ができるので、スプリント計画の精度が向上するのでリソースの最適配分が可能になります。また、プラットフォームごとの相違点を理解することで、実装上におけるリスクの早期特定から適切な対策を立てることができ、スプリント中に問題が顕在化する前に対処が可能になるので、結果としてスムーズな開発進行を支援する事にも繋がっていきます。

本稿で提示するのはほんの一例にはなりますが、

- 各プラットフォームのガイドラインやベストプラクティスにできるだけ従う- 同じ機能でもプラットフォームによって実装アプローチが異なる場合がある点に注意する- ユーザー体験を損なわないよう、各プラットフォームの特性を理解した上で設計・実装する

という観点から判断をする際の参考になれば嬉しく思います。

【普段から心がけておくと良いポイント】

1. 定期的な公式ドキュメントの確認・新機能やデザインの変更点の共有
2. 両プラットフォームに共通する部分と独自に調整すべき部分の明確化
3. プロトタイプ段階でアニメーション・インタラクションの確認
4. iOS/AndroidのUI処理における相違点やアプリケーションとの調和の確認
5. 定期的なデザインレビューやユーザビリティテストの実施
6. 各プラットフォームにおける

実装の課題や成功事例の共有

1. iOS/Android 間における差異を意識した UI 表現例

iOS/Android間で考え方方が異なる事例（1）

※iOSはUIKit / AndroidはXMLでのLayoutを想定した場合を考えています。

アスペクト比に合わせた形

iOS: UICollectionViewを利用した実装

UICollectionViewを利用する場合においては、この様な方針を取る事がが多い。

- ① UICollectionViewLayoutを利用する（Layout属性設定をカスタマイズする）
- ② UICollectionViewCompositionalLayoutを利用する（Grid表示に沿う様なレイアウト定義）

計算しやすくする1つの方法としては写真のアスペクト比をJSONで予め持っておく等…

```
{
  "id": 1, "title": "タイトルが入ります。", "summary": "概要が入ります。",
  "image": {
    "url": "...(画像のURL)...", "width": 425, "height": 640
  },
  "gift": { "flag": false, "price": null }
}
```

※ サムネイル高さ = サムネイル幅 * height / width

Android: RecyclerViewを利用した実装

RecyclerViewを利用する場合においては、StaggeredGridLayoutManagerを利用する。

```
val staggeredGridLayoutManager
    = StaggeredGridLayoutManager(2, StaggeredGridLayoutManager.VERTICAL)
```

recyclerView.layoutManager = staggeredGridLayoutManager

※ 縦方向のGrid表示

- ① iOS/Android間におけるレイアウト構成方針と特徴を把握することがまずは大切なポイント
- ② 実際に不安な場合は事前に簡単な形で試してみて、その結果を元に実現可能な形を模索すると良い

▲図 13.1 iOS/Android 間で考え方方が異なる事例（1）

iOS/Android間で考え方方が異なる事例（2）

※iOSはUIKit / AndroidはXMLでのLayoutを想定した場合を考えています。

Tab要素とContentが連動

iOS: UICollectionView (UIScrollView) とUIPageViewControllerの組み合わせ

2つの要素を別々に表示して1つの画面内で連動させる様なイメージで考える。

- ① コンテンツ表示部分（UIPageViewControllerで作成された一覧表示部分）に関する処理概要
- UIPageViewControllerDelegateの `didFinishAnimating` 处理時にIndex値に応じてタブ位置を更新する。

※ タブ型表示部分のIndex値を合わせる計算が必要
- ② タブ型表示部分（UICollectionView or UIScrollViewで並べた要素一覧部分）に関する処理概要

```
func scrollViewDidScroll(_ scrollView: UIScrollView)
```

Tab要素下にある下線表示部分がコンテンツ表示と連動して動く様な感じの処理をする。
X軸方向のOffset値を計算してコンテンツ部分のScroll変化量と合わせる様な形にする。
任意のTab要素を押下すると、動く方向を考慮して該当コンテンツ要素へ移動する。

Android: ViewPagerとTabLayoutの組み合わせ

純正Componentを応用する事で実現可能 (`ViewPager`が`UIPageViewController`に相当)

公式のドキュメントでも実装方針や方法が示されている: [iOSで言う所の何に該当するか？という発想](https://developer.android.com/guide/navigation/navigation-swipe-view?hl=ja)
<https://developer.android.com/guide/navigation/navigation-swipe-view?hl=ja>

iOS/Androidで提供されている形を知っておくと便利な事が多い:

- ① Apple純正のUI関連部品の標準の見た目や機能に注目して実装しやすい形を選択する
- ② どちらかのOSに無理に合わせていく方針よりも「そのOSにとって自然な形」の方針を取る方が良い

▲図 13.2 iOS/Android 間で考え方方が異なる事例（2）

2. プラットフォームで提供しているUI構成要素に関する例



▲図13.3 iOSでは用意されていないがAndroidでは用意されている表現例

3. 宣言的UIを例にした考え方を合わせられる可能性があり得る例

複雑な要素を伴う画面を宣言的UIで考え方を合わせる

そのまま考えるとかなり大変そうなレイアウトに見えるが、見方を変えてみると考えやすくなる場合もあります。
※iOSはSwiftUI / AndroidはJetpack ComposeでのLayoutを想定した場合を考えています。

大きな正方形要素と小さな正方形要素が規則的に並んでいる形

① "1 section has 3 items." の考え方
この様に1つのSection単位で分割すると考えやすくなる。
部品構造自体もSwiftUI・Jetpack Composeを利用する事で直感的に組み立てられる余地もあるかと思います。
② 3個入りでLayoutが異なる3パターンのSectionと捉える。
(最悪3個で1つのCell要素と考えるのもOK)
※ Section表示を繰り返し表示している発想に近い

③ この様に1つの要素内に6つの部品があると考えるイメージを持つと良い。
一覧表示をする処理の中でPaginationを伴う場合が多いですが、
iOS/Android間で考え方方が異なる点にも注意が必要です。

↓
旧来の実装だと難しそう？
iOS/Androidの旧来の考え方だと結構難しいUI実装に見えるが、宣言的UIだとシンプルにできる場合もある。

SwiftUI	Jetpack Compose
HStack (with Padding) - AsyncImage (左側の大きな画像)	ConstraintLayout - AsyncImage (左側の大きな画像)
VStack (with Padding) - AsyncImage (右上側の小さな画像)	- HorizontalDivider (縦線をPaddingに合わせて)
- AsyncImage (右下側の小さな画像)	- AsyncImage (右上側の小さな画像)
- Divider (横線をPaddingに合わせて)	- Divider (横線をPaddingに合わせて)

▲図13.4 複雑な要素を伴う画面を宣言的UIで考え方を合わせる

4. 端末固有の機能を考える場合の例



▲図 13.5 動画プレイヤー機能など端末に依存するものは特に注意が必要

アジャイル開発の観点から iOS/Android 間の差異を押さえることで、開発プロセスが大きく改善されます。プラットフォーム固有の特性を理解することで、迅速な開発と反復が可能になり、潜在的なリスクを早期に特定し管理する事にも繋がると思います。

また、効率的なリソース配分と品質向上につながり、各プラットフォームのベストプラクティスに基づいた最適な実装が実現します。これにより、ユーザー体験の一貫性を保ちつつ、各 OS の特性を活かした付加価値を提供でき、顧客満足度の向上につながります。さらに、チーム内のナレッジ共有が促進され、開発者のスキル向上と柔軟な対応力の獲得が期待できます。

13.5 なぜこの観点がアジャイル開発において重要だと考えるか？

「デザインから逆算して難易度を見積もるための観点」がアジャイル開発において重要となる理由は、アジャイル開発の特性や目指す成果と密接に関係するからだと考えています。

1. 迅速なフィードバックサイクルの促進

アジャイル開発では、短いスプリントで継続的にプロダクトをリリースし、ユーザーからのフィードバックを迅速に反映することが求められます。デザインから逆算して難易度を正確に見積もることで、スプリントごとに達成可能なタスクを正確に割り当てることができ、フィードバックサイクルを円滑に進めることができます。

2. 優先順位の明確化とスコープ管理

アジャイル開発では、機能やタスクの優先順位付けが重要です。デザインから逆算して難易度を見積もることで、技術的に複雑で時間のかかるタスクを早期に把握し、スプリントの計画段階で

適切に優先順位をつけることができます。また、スコープ管理がしやすくなり、リソースを最適に配分することで、開発の進行をスムーズに保つことができます。

3. チームの共通理解と連携の強化

アジャイル開発は、チーム全体の連携と協力が非常に重要です。デザインから逆算して難易度を見積もる際、UIコンポーネントの複雑さやプラットフォーム間の差異、データフローの管理など、各メンバーが直面する課題を共有することで、チーム全体が共通の理解を持つことができます。これにより、役割分担が明確になり、チーム全体で効率的に作業を進めることができます。

4. リスク管理と技術的負債の抑制

アジャイル開発では、技術的負債を最小限に抑えることが重要です。デザインから逆算して難易度を見積もることで、潜在的なリスクを早期に特定し、適切な対策を講じることができます。これにより、スプリント後半での問題発生や、リリース直前での修正が必要になる状況を防ぐことができ、スムーズな開発を維持できます。

5. 持続可能なペースの確保

アジャイル開発では、持続可能な開発ペースを維持することが求められます。デザインから逆算して難易度を見積もることで、開発チームが過剰な負荷をかけられることなく、適切なペースで進行できるように計画を立てることができます。これにより、チームメンバーのバーンアウトを防ぎ、長期的に高い生産性を維持することができます。

6. ユーザー価値の最大化

アジャイル開発は、ユーザーにとって最も価値のある機能を優先して提供することを目指します。デザインから逆算して難易度を見積もることで、どの機能がユーザーにとって重要か、またその機能を実現するために必要な工数やリソースを正確に把握できます。これにより、限られた時間とリソースの中で最大のユーザー価値を提供するための判断が容易になります。

アジャイル開発において、デザインから逆算して難易度を見積もる観点は、スムーズなスプリント計画、リスクの早期発見、チーム全体の連携、持続可能な開発ペースの維持、そしてユーザー価値の最大化に直結します。これらはアジャイル開発の成功に欠かせない要素であり、結果としてプロジェクト全体の効率と品質を高めることに繋がるからです。

13.6 まとめ

繰り返しになりますが、モバイルアプリ開発プロセスにおいて、デザインから逆算して難易度を適切に見積もるためにには、以下の観点が重要だと考えています。

チーム全体の連携

プロダクトマネージャー・デザイナー・エンジニア・QA等、各役割が1つのチームとして協力することが不可欠になります。特にUI実装と機能ロジックは切り離して考える事が難しい場合も珍しくないため、役割や職種を越えて共通理解を持っている状態が望ましいと思います。

技術的な実現可能性の評価

デザインが提案された時点で「実現が技術的にどれほど難しいか？」をという問い合わせ常に持ち、評価する事も重要になってきます。特に、デザインができる限り完璧に再現するに当たり、複雑な実装が必要となる箇所を早期に特定する事で、実現可能性の模索や難しい場合における代替案を検討がしやすくなると思います。

時間とリソース制約の考慮

時間やリソースには限りがあるため、実装の難易度が高い部分を早期に見極めることが求められます。これにより、開発プロセスの初期段階でリスクを管理し、優先順位をつけて進めることができになると思います。

これらの観点をもとに、開発初期の段階でチームが一丸となってデザインと技術的な要件のバランスを取りながら、最適なアプローチを見出すことがアジャイルを進めていく上では重要になります。

加えて、モバイルアプリ開発において、デザインから逆算して難易度を見積もる際は、前段階での準備力が極めて重要になると考えています。機能実現において、見た目だけでなく、裏側の技術的な実現可能性を早期に評価する必要があります。時間やリソースが限られた状況で、技術的に難しい部分やリスクの高い部分を特定し、対応するための準備を進めることが重要です。この「目に見えない部分」への準備力が、素早く正解を見出すための基盤となります。

これらの準備がしっかりとできていれば、アジャイル開発の中で、デザインから逆算し、実現可能なスケジュールやリソースを確度が高く見積もることができ、チームとしての成功にも大きく繋がると思います。

13.7 参考資料

iOS/Android 間での差異を意識しながらより良い実装を考えるためのアプローチ事例については、私自身も業務内外で取り組んでいる最中です。これまで私が執筆した記事や登壇資料を何点か掲載しますので、今後の参考になれば幸いです。

- 円滑な UI & 機能実装やデザイナーとの共同作業を進めるために心がけてきた事^{*1}
- 複雑な UI 実装の壁を越えるための考え方事例紹介 (iOS/Android 間で実装を合わせるヒント)^{*2}

^{*1} https://github.com/fumiyasac/iosdc2021_pamphlet_manuscript/blob/main/manuscript.md

^{*2} https://github.com/fumiyasac/iosdc2021_pamphlet_manuscript/blob/main/manuscript.md

第 14 章

インセプションデッキからはじめる アジャイル入門

おたけ@otkshol

14.1 はじめに

本章では、朝会などの定期的に開催されている開発イベントの目的を答えることができなかつたことをきっかけに、アジャイル開発と初めて向き合った話を共有します。

インセプションデッキという言葉も知らなかった状態から、チームに必要なことを考え、カイゼンをスタートさせることができました。筆者と同じようにアジャイルに初めて向き合い、初めて悩んでいる方にとっての何かの参考になれば幸いです。

14.2 開発チームに新規着任

昨年、私はプロダクトのバッチ開発担当から別の新しい開発チームにアサインされました。5名のチームに対して、自分を含めた3名の新規着任、サブリーダー相当の方が1名離任するという規模の大きいメンバー入れ替えが発生したタイミングでの着任になります。

新しい開発チームでは「スクラムガイド」^{*1}にあるスクラムイベントに相当する開発イベントが開催されており、「しっかりアジャイルっぽいことをやっているチームすごいな。」というのが正直な感想でした。

初めての案件実施時には、疑問に思ったことをSlackに書き込むと、答えられる人に回答をすぐもらったり、必要に応じて通話を繋いでサクッと解決したり、しかもその動きをチーム全員で補い合っており、「なんて良いチームに来たんだ！」と感じていました。

しかし、上司は「メンバーの大量入れ替えに伴い、チームの文化とスタンス部分を心配している」と言っていました。チームに特に問題を感じていない自分は、その言葉の真意を理解することができませんでした。なお、表14.1は、筆者の開発チームでの開発イベントの呼び方を、一般的なスクラムイベントの呼び方に対応させた表です。呼び方そのものは本質的なものではないかもしれませんのが、その目的が必ずしも共有、一致していなかったのかもしれません。

^{*1} <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Japanese.pdf>

▼表 14.1 スクラムイベントと開発イベントの対応

スクラムイベント	私の開発チームでの呼び方
スプリントプランニング	開発見積もり会
デイリースクラム	朝会
スプリントレビュー	レビュー会
スプリントレトロスペクティブ	開発ふりかえり

14.3 朝会の目的...?

ある日、いつも通り開催されている朝会に上司がふらっとやってきて、一通り終わった後に「朝会ってなんでやっているんだっけ？ もしかしたらもう少し短くできるかも」と筆者とチームリーダーに質問しました。しかし、筆者もチームリーダーも、その質問に明確に答えることはできませんでした。

この質問に答えられないということは、毎日行われている朝会は実施されているが、何のために行っているのか誰も分からぬ恐ろしい状況であることを、このタイミングで初めて認識しました。

朝会は日々の作業内容を報告すれば良いのかなとぼんやり思っていましたが、進捗状況を共有する理由、朝会では話すべき話題など、会議体として重要な項目が明確でないまま運用されている現状がそこにはありました。

目的や内容がはっきりしていないので、作業進捗共有の流れから話題が逸れて議論が発散してしまい、朝会そのものに時間がかかるってしまうことも多々ありました。朝会は開発メンバー全員が出席しており、かなり多くの工数を割いています。筆者には、本当にかけた工数の価値がある時間になっているか、自信を持って YES と答えることができませんでした。

14.4 インセプションデッキとの出会い

このままでは良くないことは理解でしたが、どうやって現状を打破できるか途方に暮れていきました。そんな自分に対して、上司は「インセプションデッキ的な何かをチームで取り組んだ方が良いかもね」という助言をくれました。その助言を聞いた自分は「なるほど、わからん」状態でしたので、その言葉の意味を調べ、書籍「アジャイルサムライ-達人開発者への道」(オーム社、2011) にたどり着きました。

インセプションデッキ^{*2}とは 10 個の質問から構成されるプロジェクト開始前に認識を合わせておきたい内容で、5 つの「Why」と 5 つの「How」の合計 10 個の質問から構成されるものです。主にプロジェクトチーム発足時に使用されるのですが、多くのチームメンバーが入れ替わったチームにも適用可能だと思いました。

しかし、10 個の質問すべてに取り組もうとすると膨大な時間がかかり、今のチームには必ずしも取り組む必要がない質問もあると感じました。そこで、チームリーダーともインセプションデッキの実施とその内容を議論を行いました。その結果、開発チームのキックオフを対面開催し、その

^{*2} インセプションデッキのテンプレート <https://github.com/agile-samurai-ja/support/tree/master/blank-inception-deck>

中でインセプションデッキの1つである「我々はなぜここにいるのか？」を取り組むことを決めました。

▼表 14.2 インセプションデッキ 10 個の質問

1. 我々はなぜここにいるのか？
2. エレベーターピッチ
3. パッケージデザイン
4. やらないことリスト
5. 「ご近所さん」を探せ
6. 技術的な解決案を描く
7. 夜も眠れない問題
8. 期間を見極める
9. トレードオフスライダー
10. 何がどれだけ必要か

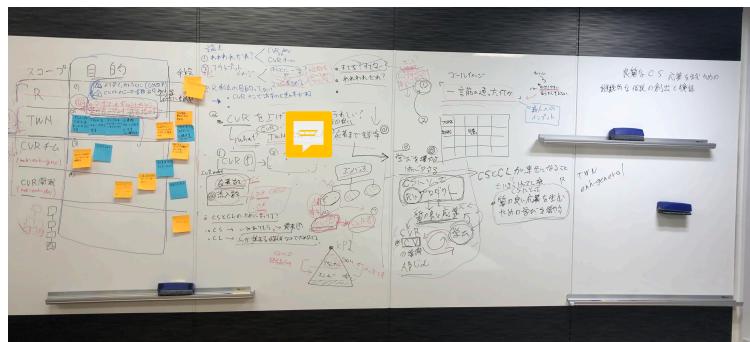
14.5 チーム対面キックオフと得られたもの

事前に出社日を調整して開発チームキックオフを対面開催しました。午前中はキックオフを行い、午後から「我々はなぜここにいるのか」を開発チームで実施しました。議論は白熱して、半日以上の時間をかけてメンバー全員で合意した回答とまとめることができました。

「我々はなぜここにいるのか」に答えることによって、開発メンバー全員で合意したチーム目標が出来上がり、新しい施策や既存の施策の目的を議論するときのチームが立ち返る場所ができました。

個人的な収穫もありました。「我々はなぜここにいるのか」を議論する際、やったことのないファシリテーション役に挑戦してみましたが、意外とやってみれば何とかなることを学びました。同時に、悩んでいる様子や喜んでいる様子が表に出過ぎる性質があるので、ファシリテーションをするときは議論の進める邪魔にならない程度に、もう少し感情コントロールができるとより良いという課題も自覚することができました。普段の業務でファシリテーションスキルと向き合う機会があまりなかったので、貴重な経験になりました。

そして、初めて対面したチームメンバーとお昼に美味しいラーメンと一緒に食べた絆も手に入れることができました（嬉しそうに卵をためらうことなくトッピングしていつもより豪華に）。



▲図 14.1 キックオフ時のホワイトボードの様子

14.6 キックオフのその後

インセプションデッキのおかげで、開発メンバー全員でチーム目標を決めることができました。しかし、これはスタート地点に立ったに過ぎません。

現在は、いつの間にか目的が明確でなくなってしまった定例イベントの目的を開発チーム振り返りの時間を使って少しずつ見直しを進めています。今まで当たり前のように過ごしていたイベントの目的を改めて考えるので、そう簡単に議論が進まないのが正直なところです。

しかし、今の我々には立ち返るチーム目標があるので、議論が発散しそうになったり、判断に迷った時はチーム目標に立ち返り、着実に議論を進めることができるようにになりました。これは大きな一歩なのではないかと思います。

チーム全員が定例開発イベントの目的が言えること、その目的がメンバー間で相違がないこと、必要に応じて柔軟にアップデートできることを目指して今日もカイゼンに取り組んで参ります。

14.7 まとめ

今回の出来事で初めてアジャイル開発と向き合い、目の前のチームに必要なものは何かを考え、もがきながら実践する貴重な経験をすることができました。次は自分が文化を吹き込んで開発チームのアウトカムを最大化させて、プロダクトを使っている方々、チームメイトの役に立てるようになります。

もし、自分と同じようになんとなく朝会など定期イベントは実施しているけど「何でやっている?」という質問に答えられない、あるいはチームメンバーで回答バラバラな状況であれば、まずはインセプションデッキの「我々はなぜここにいるのか」だけでも時間をかけて考えることをオススメします。チームの拠り所ができて、以降の議論がスムーズになると思います。



竹内尊紀 @otkshol <https://twitter.com/otkshol>

株式会社リクルートに新卒入社。HR領域の開発を主に担当し、現在はタウンワークの開発に携わっている。

第 15 章

独学でアジャイルを学ぶ

渡部 啓太 @sobarecord

15.1 独りで学ぶという選択肢

ここまで読み進めてきて、「アジャイルの実例はわかった、情熱もある。ただ、実践の場が無くて……」。そんな方もいるかもしれません。

社内でアジャイルに関する研修の講師をしていると、同じような声をよく聞きます。「アジャイルで開発をやってみたいんです。だけど、今いるプロジェクト（あるいは部署、チーム）では既存の開発手法から変わらなさそうなんですよね」。

これでは、アジャイルに習熟していくためのフィードバックループが回りません。同じように、もどかしさや機会のなさに嘆いている方は多いのではないでしょうか？

うーん、残念。でも、あきらめる必要はありません。独りで学びましょう。独りで「スクラム」をするのです。

あたりまえですが、独りだとチームで取り組むことはできません。なので、正確にはスクラムとは言えません。しかし、独りでできる範囲であれば、経験を積めます。学習が、回ります。独りでスクラムをしても、アジャイルに関する経験値はあがります。

スクラムに関する本を読んだとき、あるいはカンファレンスで他社事例を聞いたとき、独学の効果はあらわれます。自身の経験をふまえた解釈や理解ができるようになります。

筆者は日々、講師だけではなく、アジャイル開発チームの立ち上げの支援や、スクラムマスターとしての仕事もしています。成長していくチームメンバーを見ていて感じるのは、「アジャイルは体験してみないと、真の理解ができない」ということです。アジャイルソフトウェア開発の定義である「アジャイルソフトウェア開発宣言」に具体的なやり方は示されていません。自分でうまいやり方を見つけていく必要があるのです。

独習アジャイルをしましょう。

15.2 独習アジャイルの進め方

ここから、独習アジャイルの具体的な進め方を紹介します。独習アジャイルでは、アジャイルを実践するためのフレームワークであるスクラムを採用しています。

スクラムに関する用語には括弧をつけています。それぞれの用語の内容を知っているもの

として説明しているため、もしわからぬ単語があれば「スクラムガイド」や初学者向けの本（「SCRUM BOOT CAMP THE BOOK[増補改訂版] スクラムチームではじめるアジャイル開発」（翔泳社、2020）や『アジャイル開発とスクラム 第2版 顧客・技術・経営をつなぐ協調的ソフトウェア開発マネジメント』（翔泳社、2021）がおすすめ）を参照してください。「スクラムガイド」は、インターネットで検索すれば、すぐにでてきます。

スクラムをするといつても、独りなのでそこまで構える必要はありません。誰かの承認も、上長への報告義務もありません。やることも簡単です。もしかすると、すでに仕事のやり方として身についている部分もあるかもしれません。ただ、そこに「スクラムをやっているんだ」という意識をのせていきましょう。

15.2.1 準備をする

まずは準備です。自分が持っているタスクをひたすら洗い出しましょう。これは仕事のことのみでもよいのですが、プライベートなタスクも含めると、よりスクラムの効果を実感できると思います。自分の頭の中にあるすべてを外に出しましょう。目に見える形にすることで、「透明性」が上がり、自分の状況を認知しやすくなります。

注意点は1つ。全てのタスクを1日以下の大きさにしましょう。それ以上の大きなタスクだと、日々の状況を「検査」（望ましくない変化や問題の検知）することができませんからね。

洗い出しに使うのは紙のノートでもいいですし、TrelloやMiro、NotionといったWebサービスを駆使してもいいです。あるいは、パソコン上のテキストファイルやExcelでもOK。自分の使いやすいツールを使いましょう。

タスクを洗い出せたら、それらを優先順に並び替えましょう。期日や重要度、タスクが生みだす価値、今後のリスクを基準として順番を入れ替えていくとよいでしょう。このリストが、「プロダクトバックログ」です。

特別な事情がない限り、1週間を一区切りとして「スプリント」を回していくのがわかりやすいです。たとえば、月曜始まり日曜終わりのような、くり返しやすいタイムボックスを設定しましょう。

15.2.2 「スプリント」を過ごす

「スプリントプランニング」

さて、準備が整ったのなら、いよいよ独りスクラムの始まりです。次の1週間でやることを「プロダクトバックログ」から取り出しましょう。もちろん、優先順位の高いタスクからです。

取り出す際には、1週間でギリギリ収まる量を見極めようとする気持ちが重要です。結果として過不足があったとしても、ギリギリを見積もるチャレンジを繰り返すうちに、自分が1週間でこなせる限界量がわかってきます。だんだんと、計画作りがうまくなっています。

次の1週間でやることを計画する。これが「スプリントプランニング」です。

「デイリースクラム」

「スプリントプランニング」で「スプリント」の計画を立てることができました。つぎは、タスクをこなしていきましょう。そして毎日、決まった時間に、計画通りに進んでいるのかをチェックします。昨日は何をやったか、今日は何をやるか、何かリスクや問題はないか。

順調に進んでいるのなら問題ないですが、予定とずれてしまうこともあります。進捗が遅れている場合はリカバリー・プランを考えます。タスク内容によってはチームメンバーに相談したり、誰かの助けを借りることもあるでしょう。

このように、日々状況を「検査」し、必要なら計画を立て直し「適応」していきます。これが「デイリースクラム」です。個人的には、一日の計画を立てやすいので、朝に実施することをおすすめします。

「スプリントレビュー」

そして、1週間のおわりに成果（作った資料やもの、ソフトウェア）を他の人に見てもらいます。できればチームメンバーやマネージャからコメントをもらいましょう。得られたフィードバックから新たなタスクを「プロダクトバックログ」に追加したり、全体的な「プロダクトバックログ」の方針を再検討します。人に見てもらうのが難しい場合は、自己評価でも問題ないです。その場合は、できるだけ客観的に確認するようにしましょう。

これが「スプリントレビュー」です。

「スプリントレトロスペクティブ」

最後に、1週間のふりかえりをします。うまくいったことと、いかなかつたこと。それはなぜなのか。次の1週間をより良くするためのアクションはなにか。KPT（良かったこと、困ったこと、次に試すこと）やYWT（やったこと、わかったこと、次にやること）といったふりかえりのフレームワークを使うのも手です。

採用したアクションアイテムは、必要ならば「プロダクトバックログ」に追加しましょう。タスクではなく、自分に課すルールや心構えであれば、壁やモニターに貼りだしておきましょう。

これが「スプリントレトロスペクティブ」です。

15.2.3 スプリントを繰り返し、進化／深化させる

ふりかえりが終われば、新しい1週間のはじまりです。再び「スプリントプランニング」から始めていきましょう。

「スプリント」を繰り返していくと、タスクの増減や優先順位の変化が日々起こるはずです。「プロダクトバックログ」は常にアップデートし、最新の状況を反映しているリストにしましょう。

スクラムの基本的な流れに加え、自分がやってみたいと思うアジャイルに関するプラクティスを試していくのも、もちろんOKです。例えば、テスト駆動開発をしたいのであれば、ローカルの開発環境だけで動くようなテストやCI環境を用意すれば、誰にも文句は言われません。

「スプリント」を重ねるにつれ、リズム良く仕事をする大切さがわかってきます。毎日、毎週計画を立てる。ふりかえりをする。学習のループが回る。スクラムが機能するための三本柱である「透明性」「検査」「適応」が実感できるのです。

15.3 つまり、仕事の進め方なんです

ここまで、独習アジャイルの進め方について説明をしてきました。「なんだよ、そんなことか」と思う方もいるかもしれません。それもそのはず。ぼくの考えでは、スクラム（そしてしばしば比較対象に挙がるウォーターフォール開発）は、タスク管理の手法であるからです。

タスク管理の本を読んでいると、アジャイルの文脈で書かれていてもおかしくない文章が出てきます。

豊かではあるものの、変化の激しい現代において、ゆとりをもって無理なく結果を出していくには、今までの考え方や仕事のやり方では歯が立たない。今までとはまったく違う、新しい手法やテクニック、新しい習慣が求められているのだ（『全面改訂版 はじめての GTD ストレスフリーの整理術』（二見書房、2015）より）

「1つのことに集中し、それが完了してから次に進む」これが成功の王道（『仕事に追われない仕事術 マニャーナの法則』（ディスカヴァー・トゥエンティワン、2024）より）

小さな失敗を重ねることで、何がうまくいくか、うまくいかないかを知ることができます（書籍『「やること地獄」を終わらせるタスク管理「超」入門』（星海社、2019）より）

つまり、「仕事をうまくやりましょう」ということ。スクラムは仕事の進め方そのものなのです。独習アジャイルを進めていけば、アジャイルの重要な点はツールや技術だけではなく、仕事を進めるための考え方であることがわかるはずです。

15.4 アジャイルを説明できるようになる

最後に、独習アジャイルの重要なポイントをもうひとつ紹介します。それは「アジャイルを他者に説明できるようになる」です。

筆者が人生で初めてスクラムをチームに導入できたきっかけは、実はこの独りで行うスクラムだったのです。独習アジャイルを通じ、アジャイルの進め方とメリットを理解していました。そして、「アジャイルをやりたいという熱量」にプラスして、「スクラムで生産性があがった、働きやすくなった」という話をマネージャーに説きました。その説得の場で、「やってみよう」という返事をもらうことができました。

その後は、スクラムマスターとして、自身が学んできたこと（独習アジャイルや、それを通じて解像度が上がった本やコミュニティの情報など）をチームに共有し、スクラムマスターとしての一歩を踏み出しました。さらに、そのスクラムチーム立ち上げの経験をもとに、全社にアジャイルを推進する組織を立ち上げることもできました。そして、今ではアジャイルコーチとして様々な社内外のチームを支援しています。

独りで学べること、練習できることは多いです。スポーツであれば、座学はもちろん、筋トレや素振りが独習に該当するでしょう。アジャイルにおける筋トレや素振りが、独習アジャイルなのです。

アジャイルを学ぶ第一歩として、独習アジャイルという選択はいかがでしょうか。



渡部啓太 @sobarecord <https://twitter.com/sobarecord>

チーム設計師

チーム作りの専門家、アジャイルコーチ。チーム作りを通じ、誰もが楽しく働く社会を目指す。ソフトウェア開発者としてキャリアをスタート。より良い仕事の進め方を模索する中でチーム作りの大切さに目覚め、スクラムの導入や全社的なアジャイル推進活動を経験し、現在は NRI bit Labs に所属。アジャイルの導入支援やコーチングを行っている。チームの立ち上げや、軌道

修正が得意。

第 16 章

アジャイルの始め方～「書籍の探し方」 ～

ammonite_404

16.1 はじめに

筆者は WF(Waterfall) の経験が長いため、初めてアジャイルに触れたとき書籍探しに四苦八苦しました。本章ではアジャイルを始めたい、始めた方向けに、筆者なりの書籍の探し方を共有できればと思います。

16.2 書籍の探し方

試行錯誤の結果、わたしは下記 4 つのアンテナの張り方＝書籍の探し方にたどり着きました。わたしはこの中で 2~4 を実行しました。(1 は URL だったり、英語論文だったりして難易度が高いものもあるため)

1. 1 冊目の巻末の関連書籍を読みあさる
2. コミュニティやイベントに参加する
3. Amazon などで表示される「おすすめ本」を買いあさる
4. 1 人の著者の本を徹底的に追う

1~4 の内容を順番に紹介します。

1：1 冊目の巻末の関連書籍を読みあさる

1 冊目に読んだ本の巻末にまとめられている関連書籍を読むことで、知識を既知の書籍をもとに水平方向へ広げていく方法です。

メリット：

1 冊目から得た知識を補強できる

デメリット：

たくさんの関連書籍の中でどれが自分のレベルに合っている書籍か判定が難しい
英語論文もある

2：コミュニティやイベントに参加する

コミュニティやイベントに参加して、LT(ライトニングトーク)などからおすすめの書籍の情報を得る手法で、まったく知らない別軸へ知識を広げることもできる方法です。¹

メリット：

話題になっている書籍がわかる

おススメ書籍コーナーを持っているコミュニティもある

デメリット：

自分のレベルに合っている書籍か判定が難しい（話している方はバックボーンありきで話題に出しているため）

3：Amazon などで表示される「おすすめ本」を買いあさる

Amazon などでは、おすすめ本が表示されます。有名な書籍も紹介されるので、基礎的な知識の補強と知識を水平方向へ広げる方法です。

メリット：

幅広く関連知識を探ることができます

デメリット：

書籍の取扱が難しい（気になる書籍をすべてカートに入れた後、請求額に震える）

4：1人の著者の本を徹底的に追う

1人の著者を SNS などで徹底的に追うことで、知識がある角度（著者）から垂直方向へ深堀る方法です。

メリット：

その著者への理解が深まる

その著者の出版予定が分かる

著作の出版社の出版書籍一覧を調べると、出版社ごとの特色が分かる場合があり、深堀りがはかかる

どる

デメリット：

著者の研究内容、実践内容に情報が偏る可能性あり

上記4つが初心者でも可能な攻略ルートでした。個人的に実行した順番は 3 → 2 → 4 → 1 です。

まず、3 で表示されるような、界隈の「基礎的な書籍」を知っておいたほうがよいと考えました。それから 2 へ突撃、他の方の意見をうかがい、自分の解釈を発言、質疑して理解を深めます。4 は 2~3 の間に「この著者のことをもっと知りたい。どういう方だ？」と思った時に実行します。

一気に 2、4 が解決するのは「読書会イベントに参加」です。強制的に課題書籍を読み、イベントで意見・知見を共有、イベント後は Amazon などで関連書籍を探しつつふりかえりになるので、とてもありがたいイベントです。

また、2、3 は相乗効果で効果を発生します。イベントでタイトルをよく聞く書籍、Amazon などでよく連動して表示される書籍は「あ、またこの本だ。これは外せない本なのだな」とわかるよ

¹ 本来の趣旨であるコミュニティ・イベントちゃんと楽しむこと。本を探すことに集中しすぎて話を忘れちゃうと悲しいですよ。

うになります。

2のイベント参加にはもう1つ大きなメリットがあります。オンラインイベントでは「社内では使っていないツールに触れることができる」ため、新しいツールへの心理的ハードルが下がります。※セキュリティとコンプライアンスにはくれぐれも注意！

たとえばZoom、Slack、Google Meet、Discord、Miro、Mural、Notionなどのツールは会社によっては使いません。しかし、イベントで使用すると確実に経験値が上がります。

「これはMiroならすぐできる」→社内でもMiro使えるいか訊いてみようか。「オンラインミーティングなら背景画像があったほうがいい」→公式画像があるか確認してみよう。もしくは探そう。

コミュニケーションツールは、zoomのブレイクアウトルーム、録画等に似た機能がある場合が多いので、ほぼチュートリアルなしで即日から機能使用可能になります。

16.3 まとめ

以上、「書籍の探し方」について書いてみました。本章があなたがすてきな本、参考になる本に出会うお役に立てば幸いです。ご参考まで、自家製『アジャイルサムライ』を0地点とした場合の本の分布図（2021年版）を添付します。



▲図 16.1 bookmap_20220123_2

これは自身の備忘として作成しました。書籍のマッピングやメモ書きは脳内の整理にもなるので、読書=インプットだけではなく、「どんな本を読んだか」というアウトプットも大事かなと思います。



アンモ ナイト (ammonite_404)

ほぼブログの Qiita : https://qiita.com/ammonite_404

ある日アジャイルと出会い、化石なエンジニア→人類を目指して進化を始めました。社内プロジェクトでスクラムマスターを経験後、アドバンスド認定スクラムマスター（A-CSM）取得。

Qiita でアジャイル本の地図を作ったり、技術力 Up のため、QA へ挑戦したりしています。

第 17 章

コミュニティの探し方

おやかた@oyakata2438

あなたが所属しているコミュニティはありますか

どうやって探す？ 人見知りなんだけどどうしたら？ 何を話したらいいの？ たいていは杞憂です。まずは飛び込んでみましょう。

本章では、コミュニティの探し方、日ごろの立ち居振る舞いについて述べています。

本書はアジャイルに関する本ですが、この章で書いてあることは、アジャイルのコミュニティだけの話ではありません。とはいえ、本書に触れている方はアジャイルに興味がある方でしょう。まずは 18 章にあるようなアジャイル系のイベントやコミュニティに参加してみることをお勧めします。

17.1 コミュニティに参加する

参加するといっても、Slack や Discord といったオンラインスペースに登録し、会話を眺めたり、コメントしたり、スタンプを押したりするだけです。参加するためには、招待リンクをクリックしたり、登録メールを送るくらいのもので、お金もたいていの場合かかりません。勉強会やカンファレンスで参加者コミュニティのことを知ることも多いですね。イベントの案内ページに招待リンクが貼ってあったり、すでにそのコミュニティに参加している人から招待リンクを教えてもらったりすることで参加できます。

会話への参加、コメントも義務ではありません。アクティブな人より、やり取りを読んでいるだけな人、なんなら見てすらいない人のほうが多いくらいでしょう。イベントで一度参加したっきりの人も少なくありませんが、そういうものです。

17.1.1 コミュニティに入るメリット

コミュニティに所属することでいくつもメリットがあります。

わかりやすいメリットとしては以下のようないことがあります。

- 他の人との繋がりができる
- 単純に楽しい
- 次の/類似のイベントの情報が手に入る
- 雑談だって楽しい

- 相談事もできる
- 次のイベントで会える。
- リアルの友人になることも……

ある（共通の）トピックスに興味を持つ人が集まる場所というのはとても貴重です。バックグラウンドが近しいということですから、共通の話題で盛り上がることも頻繁にあります。あるあるネタで盛り上がる。今困っていることを相談する。相談してみるだけで気が楽になることもありますし、隣の人が同じ経験、さらには解決策を持っているかもしれません。ちょっとしたことを利害関係なく相談できるのがコミュニティスペースの大きなメリットです。

また、オンラインの勉強会、カンファレンスに参加すれば、コミュニティの Slack で繋がった人と会えるかもしれません。「いつも Slack でお世話になってますー」といった感じで会話のとっかかりにもなります。そして、そこから新しいイベントやつながりが生まれてくることもあります。

17.1.2 デメリットはあるか？

デメリットはないでしょうか？

コミュニティはたいていの場合、参加の度合い（読む、書き込む、反応する）は任意です。すべての投稿に目を通す必要もありませんし、都度反応する必要もありません。そして、加入も脱退も任意です。「なんか合わないなー」と思ったら抜ければよいのです。

四六時中コミュニティの Slack をチェックするとか、入り浸るようになってしまふと、むしろ依存症的な意味で黄色信号を気にした方がいいかもしれません。居心地がよいのでつい入り浸ってしまうのはとってもよくわかるところですが……。そういう意味では、コミュニティに関わることで、時間がどんどんなっていくのは困った点です。面白いオンライン／オフラインイベントがある。面白い本が紹介されてた。雑談に盛り上がった。ああ……時間がどんどんなってしまいます。困った。

少し活動に疲れたら、しばらく ROM (Read only member: 読んでるだけの人) だけ、あるいはしばらく離れてみるとよいでしょう。繰り返しますが参加、コメントは義務ではありません。離れているときは投稿を読む必要もありません。離れる（あるいは戻ること）を公言する^{*1}必要もありません。ゆるく繋がれるというのは、技術／オンラインコミュニティの大きなメリットです。

17.1.3 どうやって探す？

イベントに参加する

一番手っ取り早い参加のしかたは、勉強会やカンファレンスに参加することです。最近のカンファレンスや勉強会はたいていの場合、オンライン開催（併催）です。2019年までのようにオンサイト（オンライン）開催ばかりではありません。その結果、日本中どこでも参加できますし、あるいはリアルタイムで参加する^{*2}必要すらありません

さて、主催コミュニティのベース基地となる Slack や Discord があるときは、カンファレンスの

^{*1} コミュニティのボード（運営）に関わっているようになっているならば、他の運営メンバーに「疲れたからしばらく休む」くらいは伝えるとよいでしょうが、普通のメンバーならそれすら必要ありません。

^{*2} もちろんリアルタイムで参加するメリットもたくさんあります。登壇者に Zoom や Twitter などで直接質問を投げかける、（オンライン）懇親会で会話する、X(旧 Twitter) 実況をする、などなど。

参加者へ誘導がなされます。すでにたくさん的人がいるでしょう。雑談チャンネルやイベントのチャンネルで盛り上がっているといいですね。ここに入ることで、自動的にコミュニティに参加することができます。もちろん入ったからといって、そこにどこまで関わるかは任意です。ROMでも、雑談だけでも、あるいは運営にかかわっていくのでも、自由です。

人づてに探す

今いるコミュニティに参加している人、あるいはフォローしている人が参加しているコミュニティを探してみましょう。なんなら直接聞いてみるのが手っ取り早いでしょう。良さげなところを紹介してもらえるかもしれません。さらにその先で別のコミュニティに出会えるかもしれません。

気軽に入ってみて、ちょっと違うかも？ と思ったら抜けても、放置してもよいのです。

SNS のハッシュタグで探す

前述の二つにもつながるところはあります、イベントや関係者のつぶやきの中からハッシュタグを拾ってみるという方法もあります。カンファレンスなどのイベントにたいてい公式のハッシュタグが付いています。またそれに参加している人もハッシュタグをつけて実況していることがあります。そのハッシュタグを追うことで、関連するイベントや人の情報が手に入るでしょう。

17.2 コミュニティを作る

入りたいコミュニティがなかったら、いっそ作ってしまいましょう。

無料の Slack または Discord などに適当な名称でチームをつくり、そこを拠点に活動を始めるという手があります。幸い Slack も最初は無料です。無料だと 3 か月以内や 1 万ポストまでといった制約はありますが、1 万まで行くには結構な時間がかかります。Discord は無料ですし、メンバーの権限付与などの制御が Slack よりやりやすい面があります。他のプラットフォームでも大きな問題にはならないでしょう。

いずれにせよ、**コミュニティがない場合には、新しく作る**という選択肢があります。

ベース基地があるということが重要で、集まれる場所があると、人が集まってきたたり、会話が始まったりします。最初は人が少なくてあまり楽しくないかもしれません……。

またこのとき、コミュニティの位置づけを明確にしておくことで参加しやすくなる面があります。Not for Me と思われても参加者は増えませんが、何をやっているところかわからないというのも参加をためらう要因になります。

例えば、「IT 技術者（全員ウェルカム）」というコミュニティを作ろうとしたと考えましょう。あなたはそこに参加しようと思いますか？ IT 技術者ってなんだろう、自分が合致するだろうか？ 実は全然違う感じだったらどうしよう？ と思いませんか？

それよりも、「Web 系フロントの初心者」といったように、ある程度限定されていた方が入りやすくないですか？ これらはあくまで例ですが、ある程度具体的な参加者を想定する方がよいでしょう。ペルソナを設定すると言い換えてもいいかもしれません。ターゲットを明確にすることで、検索の精度も向上します。すでにコミュニティがあるかもしれません。

おっと、すでにコミュニティがあるから新しい類似のコミュニティを作ってはいけないという意味ではありません。同じ内容に見えても、別のコミュニティは別のコミュニティです。別物な

ので、参加者も異なり、雰囲気も異なります。気軽に作ってみて、うまくいかなければ閉鎖/放置すればよいのです。繰り返しになりますが、最初は無料で始められます。

17.3 コミュニティの居心地をよくする

居心地のよいコミュニティにはだんだん人が集まっています。少しづつ人が増えてくると、いつもいる人が出てきてゆるく雑談していたり、イベントの相談をしてたりといった雰囲気になります。今困っている問題を相談した（半分愚痴った）ところ、アドバイスがもらえたり、口にすることで整理されて自己解決したり、そういった経験はありませんか？

居心地のいいコミュニティは活動も盛んです。逆かもしれませんね。活動が盛んなところは、みんなが Gentle なので居心地がいいのかもしれません。

17.3.1 雜談する

誰かがいて、適当に反応が返ってくると嬉しいですよね。同じコミュニティにいるということは、少なからず興味やバックグラウンドが近しいということ。共通の話題もありますね。

ちょっとした雑談であっても、会話が弾んでいるということは、それだけコミュニティの活発さの指標になります。すべての雑談に絡む必要はもちろんありませんが、面白いと思ったら反応しましょう。スタンプを付けるだけでも OK です。自分の投稿にスタンプが付いたということは、読んで明確な反応をしてくれた人がいるということで、ちょっと嬉しくなります。

雑談ですからちょっとしたことでも OK。人がいて反応をしてくれるというだけで嬉しくなり、さらに活発になります。もしあなたが主催者であれば、ぜひ（疲れない程度に）反応をしてあげてください。初めて参加するような人にとっては、最初の投稿は案外ハードルの高いもの。そして投稿しようかどうか、関係ないって怒られないかしら、無視されたらどうしよう、など不安に思っている場合もあるでしょう。そういったときに優しく反応してあげられるといいですね。

Slack や Discord によって、非同期のコミュニケーションが成立しやすくなりました。電話ほどリアルタイムでなく、メールほど宛先が明確ではなく、ゆるく返事したりスタンプ押したり、気が向いたら返事したりとすることができます。

17.3.2 質問する

質問を投げてみるのも、コミュニティの活性化に役立ちます。情報漏洩にならないよう、個人情報を含まないようにといった留意点はありますが、今困ることなどを雑談チャンネルなどに投げてみると、思わぬところから反応があったりします。いいアイディアが出てくることもあります。「あるあるー」で共感が得られるかもしれません。自身の課題の言語化、内省といったメリットもありますが、人に話してみるだけで絡まっていたことが解きほぐされることもありますね。

そうでなくても、会話が増えることそれ自体がコミュニティの力になります。いつでも、ちょっとした質問でも投げていいんだ、という雰囲気が醸成できれば優勝です。

17.3.3 アンチハラスメントポリシーを定める

コミュニティが大きくなってくると、一定の確率でトラブルが生じる可能性が出てきます。そういうときのために、アンチハラスメントポリシーを定めておきましょう。行動規範という言い

方をする場合もあります。

ごくごく簡単に言えば、他者に敬意を持ち、攻撃的な言動や様々なハラスメントはやめましょう、という当たり前の宣言です。様々な勉強会やカンファレンスに表示されることが増えました。内容としては当たり前ですし、アンチハラスメントポリシーがなくとも、たいていの場合特に問題は生じません。関係者、参加者のほとんどはそういったトラブルを起こすような人ではありません。

セクハラやパワハラ、その他のハラスメント、攻撃的な言動が横行するようなコミュニティに参加したいと思う人はいませんよね。

しかし、アンチハラスメントポリシーがあったからといって、トラブルが起こらないというものでもありません。残念なことですが。

それでも、アンチハラスメントポリシーにて禁止事項を謳っておくことで、そういう行動をする人（たち）に対して、警告や排除といった対処を取りやすくなります。また、被害者にとっても「行動規範に違反する行為に遭遇した場合に通報していいのだ」、見かけたという人にとっても「通報していいのだ」と思ってもらう効果もあります。

アンチハラスメントポリシー/行動規範は自分で作ってもよいですが、他のコミュニティの行動規範を参考にするのもよいでしょう。

例えば、筆者も運営に関わっているイベント/コミュニティである「技術書同人誌博覧会」では、行動規範を公開しています。

<https://esa-pages.io/p/sharing/13039/posts/13/4c6fe5c0f58bb4fb32cd.html>

こちらを参考にしてみてください。

17.4 まとめ

コミュニティに参加する意義、そして作ってみることについて述べました。コミュニティに参加して得られるものもあります。単純に楽しい！ という点だけでも参加する価値はあります。ベース基地を見つけに行きましょう！



親方 @oyakata2438 <https://twitter.com/oyakata2438>

サークル名：親方 Project

大規模プラント向け計測センサの開発を本業としつつ、エンジニアの困ったこと、知りたいことをテーマについて技術同人誌を企画・編集しています。コミケ、技術書同人誌博覧会(技書博)、技術書典などのイベントに参加。またいくつかのカンファレンスにもスタッフとして参画中。楽しいことに首を突っ込みたい。

第 18 章

アジャイルイベントまとめ



J.K(@project_j_k)

18.1 各地のアジャイル関連イベント

現在アジャイルに関連するさまざまなイベントが日本中で開催されています。

コロナ禍の状況下でオンライン開催が難しい情勢にある昨今ではありますが、新しいオンラインカンファレンスが生まれ続けており、変化の中で新しい機会を生み出しているのも、アジャイルな人たちの特長だと感じます。

以下に、20 年前から続いているものからこれから誕生予定のものまで、カンファレンスの名称と公式ページへのリンクを並べてみました。知っているものがいくつあるか確認してみたり、イベントの雰囲気を見たり、次回イベントの情報を確認したりと、これからのご参考にと活用してみてください。

XP 祭り

<http://xpjug.com/>

2002 年に誕生。日本のレジェンドさんたちもニューカマーもわいわいするお祭り。いろんな人が集結していてとっても賑やか！ プロポーザルの登竜門のような存在でもあるとか。

Agile Japan

<https://agilejapan.jp/>

2009 年に誕生。「アジャイルマニフェスト」を翻訳した平鍋健児氏を中心に立ち上げられた、おそらく国内初めてのアジャイルカンファレンス。

歴代実行委員の方々が築いてきたからこそ今がある、伝統に支えられながら変化を続けている。

Regional Scrum Gathering Tokyo

<https://www.scrumgatheringtokyo.org/>

2010 年に誕生。日本のスクラムの熱量を牽引し続けるカンファレンス。

ここで登壇するために、一年間何を積み重ねていくのかを考えている人が多く、チケットやスポーツバー枠の完売の早さも風物詩となっている。

Innovation Sprint 2011

<http://innovationsprint.com/>

2011 年に開催。

「スクラム」の生みの親である野中郁次郎氏と育ての親とも言えるジェフ・サザーランド博士が集まった伝説のイベント。グローバルやイノベーションがキーワードとなっていた。

DevOpsDays Tokyo

<https://www.devopsdaystokyo.org/>

2012 年に誕生。

世界中で開催されている DevOpsDays の日本リージョン。桜の描かれた素敵なロゴは海外からもスピーカーにきて欲しいという想いからで、毎年 4 月に開催されている。

Scrum Fest Osaka

<https://www.scrumosaka.org/>

2019 年に誕生。

Regional Scrum Gathering Tokyo から生まれたスクフェス第 1 号。2 年目からはオンラインで全国の地域コミュニティが集うフェス形式に。

Scrum Interaction

Scrum Interaction 2022 <https://scruminc.jp/event/jp/scruminteraction/2022/> Scrum Interaction 2023 開催レポート <https://scruminc.jp/blog/4615/>

Scrum Inc. Japan 主催のカンファレンスで、2019 年にはじまり、2022, 2023 と開催されている。2019 年はスクラムの父、Dr. Jeff Sutherland、そして、スクラムの祖父、野中郁次郎先生らを招いたカンファレンス。ワークショップ型の内容となっており、Scrum@Scale を体験して帰れる仕組みが印象的でした。

Agile Tech EXPO

<https://agiletechexpo.com/>

2020 年に誕生。パンデミックにより対面で集まりたくても集まれなくなってしまった我々にオンラインで集まる機会を切り拓くべく発足し、始動。学生も社会人も国籍もないカンファレンスを目指し日々活動中。

Scrum Fest Mikawa

<https://www.scrumfestmikawa.org/>

2020 年に誕生。スクフェス第 2 号。初回から現地会場 × オンラインのハイブリッド開催で賑わう。Earlybird チケットも価格が開催地の三河をもじって、3,111 円(三川)なところに素敵なこだわりを感じる。

Scrum Fest Sapporo

<https://www.scrumfestsapporo.org/>

2020年に誕生。歴史と伝統のあるアジャイル札幌が主催で、暖かい雰囲気が大好き。お手製のお土産ボックスが最高に美味しい体験を提供してくれて、北海道に思い馳せてしまう。

ふりかえりカンファレンス

<https://www.facebook.com/hurikaerijissenkai/>

2021年に誕生。ふりかえり実践者たちが集う、学びの多いカンファレンス。録画を見るだけでも参加する価値があるかも。ふりカエルが可愛い！

アジャイル経営カンファレンス

<https://agile-keiei-conf.jp/>

2022年に開催。ビジネスアジリティを高めてスピーディな経営判断を実現する「アジャイル経営」という考え方を実現・推進するマネジメント層向けのイベント。

Lean Conference Japan

<https://lean-conference.com/>

2022年に誕生。「日本"式"の理想的経営を考える」トヨタ生産方式から生まれ世界中に影響を与えていたる Lean を日本からどう向き合っていくかを考える場として誕生した。

Scrum Fest Niigata

<https://www.scrumfestniigata.org/>

2022年に誕生。オーガナイザーさんの愛と情熱のおもてなしが現地参加者を歓喜の渦に。日本酒が好きな人にはたまらないフェス&地域です。

Scrum Fest Sendai

<https://www.scrumfestsendai.org/>

2022年に誕生。

初回から、サメのようなマスコットキャラクターが誕生した！

Scrum Fest Fukuoka

<https://www.scrumfestfukuoka.org/>

2023年に誕生。

スクフェスが遂に九州に上陸。

18.2 良いカンファレンスに出会う、ということ

良い、の定義はさまざまですが、筆者自身が良いと感じたカンファレンスは、様々な出会いがありました。例えばサーバントリーダーシップという言葉に初めて出会ったのも、日本にいたら絶対に会えないような海外のスピーカーのお話を聞けたのも、社内で悩んでいることを相談できる社外の仲間と出会えたのも、一緒にカンファレンスを運営してくれる・したいと思える仲間と出会えたのも、新しい職場を見つけたのも、全てカンファレンスをきっかけとして起こった出会いでした。

良い出会いは、必ず自分の人生を変えてくれました。一つ一つの出会いがなければ、間違いなく今の自分はいませんでした。この本に携わることができたのも、コミュニティからの出会いのおかげでした。

同じカンファレンスに通うと、一年越しの再開に感動することもしばしば。お互いがお互いの場所で取り組んできたことを持ち寄って、また新しい元気や勇気をもらえるのもカンファレンスの魅力です。

18.3 他にもたくさんあります

カンファレンス以外にも、様々なイベントやコミュニティなどもあります。初めはみんな知り合いがない状態からスタートですが、その一歩を知っている人たちだからこそ、暖かく歓迎してくれる人たちがそこにはいます。情報は各イベントのプラットフォームや X(旧 Twitter) アカウントなどをフォローするのがオススメです。ステキなイベント、コミュニティ、そして人の出会いに巡り合えますように!



J.K @project_J_K https://twitter.com/project_J_K

Agile Tech EXPO Organizer / Agile Japan 実行委員

アジャイルで日本から世界を楽しく！ アジャイルコーチや組織開発に従事。カンファレンス運営などを通じ、Agile が楽しく広まることを夢見て日々活動中。

第 19 章

スクラムマスターの資格の選び方

増田謙太郎@scrummasudar

仕事や技術コミュニティの活動をしている中で、スクラムマスターの資格について質問をいたることがあります。例えば、「認定スクラムマスター」という同じ名前の資格があるけれど、どの資格を取得すればよいのか?」、「研修を実施している会社が複数あるけれども、選び方がわからない。」といった内容です。

本章では、複数あるスクラムマスターの資格に関する詳細および、私の考える資格と研修の選び方について、お伝えします。

19.1 そもそもスクラムマスターに資格は必要?

スクラムマスターになるために、資格は必要ありません。医師のような業務独占資格、キャリアコンサルタントのような名称独占資格ではありません。そのため、ソフトウェア開発を含め、ものごとをスクラムで進めていこうと決め、スクラムマスターを担当することになれば、その日からスクラムマスターを名乗ることになります。

しかし、スクラムマスターを担当することになっても、スクラム初心者の場合であれば、初日からスクラムマスターの責任を果たすことは難しいです。スクラムとはなにか、スクラムマスターとはなにかを理解した上で、スクラムチームやステークホルダーに対して、スクラムマスターの責任を果たす必要があります。

スクラムやスクラムマスターを理解するための方法として、スクラムガイドや関連する技術書を読む、会社内や技術コミュニティなどにいる先輩スクラムマスターーやアジャイルコーチに相談する、カンファレンスに参加するなどがあります。数ある方法の中で、1つの有用な方法として、「スクラムマスターの研修を受け、資格を取得する」があります。

19.2 スクラムマスターの資格について

2024年9月現在、スクラムマスターの資格は、大きく3つあります。それぞれ提供する団体・企業が異なるため、日本語で「認定スクラムマスター」と表現されていても、別の資格です。

- Scrum Alliance®が提供する認定スクラムマスター (Certified ScrumMaster® /CSM®)
- Scrum Inc. が提供する認定スクラムマスター (Registered Scrum Master®)
- Scrum.org が提供する Professional Scrum Master™

19.2.1 認定スクラムマスター (Certified ScrumMaster® /CSM®)

日本において「認定スクラムマスター」の資格で真っ先に想像するのが、Scrum Alliance®の提供する Certified ScrumMaster® (以下 CSM) です。2日間から4日間の研修を受け、その後テストに合格することで、資格を得ることができます。

2024年9月時点では、日本で研修を実施している企業は、次の5社です。研修日程は、Scrum Alliance®の Web サイト^{*1}、もしくは各社の Web サイトから確認することができます。

- アギレルゴコンサルティング株式会社
- アジャイルビジネスインスティテュート株式会社
- 株式会社アトラクタ
- 株式会社 Odd-e Japan
- TIS 株式会社

研修を受けた後、講師からテストを受ける資格があると認められると、テストに関する案内が送られてきます。テストはオンラインで実施され、テストに合格すると資格を得ることができます。資格取得のための費用は、20万円から30万円程度です。

19.2.2 認定スクラムマスター (Registered Scrum Master®)

日本語では同じ「認定スクラムマスター」ではありますが、Registered Scrum Master® (以下 RSM) は、Scrum Inc. が提供する資格です。2022年7月に、Licensed Scrum Master から Registered Scrum Master® に名称が変更されました。

2日間の研修を受け、その後テストに合格することで、資格を得ることができます。

2024年9月時点では、日本で研修を実施している企業は、次の3社です。研修日程は、各社の Web サイトから確認することができます。

- Scrum Inc. Japan 株式会社
- 株式会社永和システムマネジメント
- LSA CONSULTiNG 株式会社

資格取得のための費用は、20万円程度です。

19.2.3 Professional Scrum Master™

Professional Scrum Master™ (以下 PSM) は、Scrum.org が提供する資格です。PSM は、I から III までの段階が設定されており、I が基礎コースになっています。CSM や RSM と異なり、研修への参加が必須ではなく、テストに合格するのみで、資格を得ることができます。

また、日本語への対応が他の2資格と比較すると遅れていますが、英語のみで実施されています。そのため、英語に不安がある方でも、問題ありません。

テストを受けるための研修は不要ですが、Professional Scrum Master™ I への試験料

^{*1} <https://www.scrumalliance.org>

この研修が実施されています。2024年9月時点では、日本で研修を実施している企業は、次の2社です。研修日程は、Scrum.orgのWebサイト^{*2}、もしくは各社のWebサイトから確認することができます。

- ・株式会社 IT プレナーズジャパン・アジアパシフィック
- ・サーバントワークス株式会社

資格取得のための費用は、テストのみであれば、2万円程度です。研修を受ける場合は、20万円程度です。

19.3 資格の選び方

資格が3つあり、それぞれの差について説明をしました。その中で、どの資格を選択するべきかということになります。しかし、資格そのものに差はない、と私は考えています。むしろ、どの講師の研修を受けたかが、大事だと考えています。同じ資格であっても、講師によって研修内容が大きく異なります。研修を受ける前には、アジャイル開発やスクラムに、多少の差はある、すでに触れた状態だと思います。その状態で、どの研修が、自分にとってよいかを考えることが、重要です。

研修の選び方は、以下4つの軸があると、私は考えています。

- ・講師の考え方や理念で選ぶ
- ・講師の得意分野で選ぶ
- ・日本語で講師と対話できるかで選ぶ
- ・研修方式で選ぶ

19.3.1 講師の考え方や理念で選ぶ

講師は、書籍の執筆・翻訳に携わっている、カンファレンスで発表している場合があります。そのため、書籍や発表で感銘を受け、講師の考え方や理念を深く知りたいという場合があると考えています。

研修に参加するのは、あなたご自身なので、その講師の研修を受けたいという気持ちが非常に大事です。

例えば、『SCRUMMASTER THE BOOK 優れたスクラムマスターになるための極意——メタスキル、学習、心理、リーダーシップ』(翔泳社,2020) のZuzana Sochovaさん、『組織パターン チームの成長によりアジャイルソフトウェア開発の変革を促す』(翔泳社,2013) のJames O. Coplienさん、『SCRUM BOOT CAMP THE BOOK【増補改訂版】スクラムチームではじめるアジャイル開発』(翔泳社,2020) の吉羽 龍太郎さん、『これだけ! KPT』(すばる舎,2013) の天野 勝さんのように、書籍を執筆されている講師もいます。彼、彼女らの書籍を読んで、既に仕事で生かしている場合に、おすすめの選び方です。

研修を受ける前に、講師について知るためカンファレンスに参加し、コミュニケーションを取ってみることも、研修をより良くする事前準備になるでしょう。日本では、Regional Scrum

^{*2} <https://www.scrum.org>

Gathering Tokyo、日本各地で開催されるスクラムフェス、アジャイルジャパンなどのカンファレンスがあり、参加してみるとよいでしょう。

19.3.2 講師の得意分野で選ぶ

講師はスクラムマスターの研修ができるだけの、全般的なスキルを当然持っていますが、それでも講師それぞれが得意とする分野があります。ソフトウェア開発に関することが得意分野の方もいらっしゃれば、プロダクトマネージメントを得意とする方もいらっしゃいます。

研修に参加するあなたご自身や、関わっているチーム・組織の課題に合わせて、講師を選ぶことで、研修の体験がよりよくなると思います。

例えば、アジャイルビジネスインスティテュート株式会社の Joe Justice さんは、Management 3.0 に精通しています。スクラムマスターに関する事柄だけではなく、Management 3.0 に関する事柄を、研修を通して、体験し、学習することができると思います。LSA CONSULTiNG 株式会社の松永 広明さんは、エンベデッド(組み込み開発)領域や非 IT 領域へのアジャイル開発の経験があります。ソフトウェア開発とは離れた分野でスクラムを実践される方にとっては、適切な相談先となるでしょう。スクラムについて、通常よりも深く知りたい場合には、スクラムをパタンランゲージ形式で記述した書籍『A Scrum Book: The Spirit of the Game』(Pragmatic Bookshelf,2019) の著者である原田 騎郎さんがおすすめです。いくつかのチームや組織でスクラムマスターを経験された方は、過去うまくいった事象について、高度な言語化を体験できると思います。

19.3.3 日本語で講師と対話できるかで選ぶ

講師が英語話者の場合、通訳の方を通して研修内容を理解することになります。通訳の方はアジャイルコーチをされているなど、スクラムに精通している方が多いですが、せっかく研修に時間を使うのであれば、母国語である日本語で研修を受け、質疑応答を多くしたいという方もいらっしゃると思います。

講師と数多くのコミュニケーションを取りたいという場合は、日本語で研修をする講師を選ぶとよいです。

19.3.4 研修方式で選ぶ

コロナ禍以前は、会議室で実施されるオンサイト研修が一般的でした。研修の多くが東京で実施されており、地方で開催される研修の数は少なかったです。しかし、コロナ禍では、オンライン研修が主流になりました。コロナ禍が落ち着いた現在では、オンラインとオンサイトの研修を受講者が選択できるようになっています。

現在、研修方式は、大きく 3 つあります。

- 通常のオンサイト研修
- 合宿型のオンサイト研修
- オンライン研修

通常のオンサイト研修

通常のオンサイト研修は、コロナ禍以前からある、伝統的なスタイルです。研修が実施される会議室に参加者が集まり、講義を受けたり、ワークショップに取り組みます。実施されるオンライン研修の多くが、このパターンに当てはまります。

オンライン研修かオンライン研修のどちらかで迷った場合には、私は基本的にオンライン研修をおすすめしています。オンライン研修では、研修の間にあるランチや休憩の時間も講師や他の受講者と過ごすために、直接的な研修の時間以外でも、接点を持つことができます。そのため、講師にオンライン研修よりも質問ができたり、受講者同士で感想を共有したりすることで、研修の内容を深めることができます。また、各日の研修が終わった後、講師も参加する飲み会が深夜まで開かれることもあります。そういった、研修以外の時間を多く過ごすことで、その先も友人として交流を持ち、スクラムについてより学べる環境に繋がります。

合宿型のオンライン研修

合宿型のオンライン研修は、ホテルなどの施設で研修を受けるスタイルで、日本では近年生まれました。現在、合宿型のオンライン研修を実施しているのは、株式会社アトラクタ、株式会社永和システムマネジメントの2社です。

合宿型のオンライン研修は、研修が行われるすべての時間を講師や受講者と過ごすため、会議室で行われる通常型のオンライン研修よりも、更に濃密な時間を過ごすことができます。普段の仕事場と隔離された環境だからこそ取れるコミュニケーションが活発に行われ、より深い学びにつながります。

オンライン研修

オンライン研修は、コロナ禍に生まれたスタイルで、現在多くの会社が実施しています。オンライン研修は、研修時間と休憩時間が明確に分かれており、資格取得という観点では一番コストパフォーマンスがよいです。研修中に講師への質問時間は十分にあるため、資格取得に向けた学習という観点では不足はありません。

受講者同士でのコミュニケーションは、あくまでワークショップなど研修の一環の中で行われることが中心です。もちろん、受講者同士が活発に研修時間外でもSNSなどを通してコミュニケーションを取っている場合もありますが、オンライン研修と比較すると少ないです。受講者同士での仲を深めたい方にとっては、オンライン研修は物足りなく感じことがあるでしょう。

オンライン研修にはないオンライン研修のメリットは、言語の壁を乗り越えることができれば、世界中の研修にアクセスできることです。日本語での研修を必須としないのであれば、本章で記載している日本の会社以外にも調べてみるとよいでしょう。CSMであれば、Scrum Alliance®のWebサイト^{*3}から、世界中のオンライン研修を調べることができます。

19.3.5 他の選び方について

原則、研修は平日に実施されます。しかし、どうしても仕事の都合で、土日に研修を受けたい方もいらっしゃると思います。アジャイルビジネスインスティテュート株式会社では、土日に研修

^{*3} <https://www.scrumalliance.org>

を実施している場合があります。日程の柔軟性を優先したい場合は、アジャイルビジネスインスティテュート株式会社の研修日程を調べてみるとよいでしょう。

19.4 おわりに

本章では、スクラムマスターの資格の詳細と、資格と研修の選び方について、紹介しました。特に、2日間から4日間と長い時間を過ごす研修を、どのように有効活用するとよいかという観点で、研修の選び方を紹介しました。「講師の考え方や理念で選ぶ」、「講師の得意分野で選ぶ」、「日本語で講師と対話できるかで選ぶ」、「研修方式で選ぶ」のいずれの選び方も良い方法だと考えていますし、複数の観点をかけ合わせてより良い回を選ぶことも可能だと思います。

今後、スクラムマスターの資格を取得したいと考えている方の参考になればさいわいです。



増田 謙太郎@scrummasudar <https://twitter.com/scrummasudar>

<https://scrummasudar.com> <https://scrummasudar.hatenablog.com>

フリーランス(屋号:SCRUMMASUDAR)のスクラムマスター。スクラム道関西運営メンバー。アジャイルラジオメインパーソナリティ。2014年、セキュリティソフトウェア企業でアジャイル開発に出会い、2015年からスクラムマスターを担当。2021年、フリーランスとなり、ゲーム業界にてスクラムマスターとしてLeSSに取り組んでいる。

第 20 章

アジャイルを勉強した後のキャリアの 5 つのロールモデル

佐竹 朱衣子

20.1 最初に

アジャイルについて興味がある方々が本書を読まれると思います。これは、アジャイルを勉強した後にどんな未来があるのか、気になる方向けの記事です。自分の考えを整理するために書いた文章ですが、ぜひ読者の方にも、アジャイルを学んだ後にどんなキャリアを築けるかを考えて頂きたいと思います。

20.2 キャリアの 5 つのロールモデルを考えることになったきっかけ

私は「Chikirin の日記」^{*1}というブログのファンなのですが、この方の記事「キャリア形成における「5 つのロールモデルメソッド」^{*2}を参考にしました。この記事では、具体的にエンジニアの 5 つのロールモデルが示されており、自分のキャリアというものに無自覚でいた私に考えるきっかけをくれました。以下、記事を引用します。

「私はエンジニアの人には、キャリアのロールモデルを 5 つぐらい提示して選んでもらったらいいんじゃないかなと思ってるんです。

たとえば、第一の道として、「この分野に関してはコイツの右に出る奴はない」みたいなオタクエンジニアになる道。狭く深い知識で生きていくことになるから社内での出世は難しいけれど、ノーベル賞を取るぐらいの勢いで頑張る人なら挑戦していい道だよ、と。

2 つめに、そこまで高い技術力はもっていないけれど、トレンドに合わせて売れる商品を器用に開発していくという売れっ子エンジニアの道。

3 つめは、エンジニアとしては「まあ、ちょっとね」という人でも、マーケティングや営業をやらせると、技術のバックボーンを活かして他の営業マンとはひと味違う営業をやりま

^{*1} <https://chikirin.hatenablog.com/about>

^{*2} <https://chikirin.hatenablog.com/entry/20120512>

す、という道。

(中略)

「俺は5つのうち、どれに向いているエンジニアかな?」と考えながら必要な勉強を積んでいくことは、エンジニアのためにも、会社のためにもなりませんか?」

ぜひみなさんにも上述の記事を読んでもらいたいと思いますが、私がこれを最初に読んだのは、システムエンジニアとしての自分のキャリアについて迷っていたときでした。

当時、子どもがまだ小さく、これまでセンスがないなりに長時間使って進めてきた開発の仕事も時短勤務になり、上手く回らなくなりました。同じ時短勤務でも、時間の使い方の工夫や高い技術で効率的に進めている方もいらっしゃる中で、このままではいけないと焦ってばかりでした。

そんなときにこの記事に出会い、別のキャリアもあり得るのかと感銘を受け、方向転換を考え始めました。時間はかかりましたが、最終的に社内の開発セクションから別のセクションへの異動希望を出し、異動先でアジャイル開発を知りました。アジャイルの考え方へ感銘を受け、勉強しながら社内のアジャイル推進を行い、何とかキャリアを重ねていくことが出来ました。

このような経緯があったことから、アジャイル開発という、より狭い範囲でのキャリアのロールモデルを整理するのは面白いのでは、と考えました。

20.3 アジャイルに関するキャリアの5つのロールモデル

では、アジャイルに関わるキャリアには、どんなロールモデルがあるのでしょうか。

アジャイルは、狭義にはソフトウェア開発手法の一つであり、アジャイルを勉強したからには、アジャイル開発における開発者を目指すのがすぐに思い浮かぶモデルかもしれません。しかし、周囲のアジャイルに関わる方々を見ていると、決して道はそれだけではなく、アジャイルを手持ちの札に加えることで確実にキャリアの幅が広がると感じています。

ちきりんさんの例にならって、アジャイルに関する5つのロールモデルを上げていこうと思います。

1. アジャイル開発チームの中で輝くスーパー開発者

主にスクラムチームにおける開発者をイメージしていますが、開発チームの一員として、常に改善を心がけ、技術的負債やプロダクトのビジネスの価値についてプロダクトオーナーと対等な議論が出来る、「世界を変える」ことの出来る開発者です^{*3}。

ユーザー企業・SIer・フリーのどこでも積めるキャリアです。スーパー開発者になりたいと思ったら、まず毎日新しい情報に触れるよう習慣化することだと、見ていて思います。スーパー開発者は毎日新しい情報に触れ、試し、良いと思ったら取り入れています。毎日の積み重ねが開発者を遠いところへ連れて行くのだといつも感じています。

次に、技術はもちろんのことプロダクトへの理解を深めて、プロダクトの価値向上について考えることが重要です。プロダクトオーナーやステークホルダーに技術面での提案をする際に、プロ

^{*3} 「アウトプットは必要なものだが、本当に大事な要素は他にある。(中略) 新しくできるようになったことは何かといったものは、成果の尺度にはならない。尺度は、それを作った結果、人々がそれぞれの目的を達するために実際にやり方をかけていたり、何よりも大切なことだが、あなたが彼らの生活をより良いものに変えられたかだ。大切なのはそれだ。あなたは世界を変えたのだ。あなたは、人々が目的を達する方法を変えてしまうものを投入した。それを使うと、世界は人々にとって違うものになる。」(『ユーザーストーリーマッピング』(オンラインリージャパン,2015)より引用)

ダクトが何をどのように提供できるようになるのか、それによってプロダクトのユーザーの世界はどう変わるのがを説明できる開発者は、スクラムチーム全体にとても頼りにされています。

2. スーパースクラムマスター/アジャイルコーチ

アジャイルの深い知識も技術力も持ちつつチームを俯瞰して、チームを軌道に乗せるスクラムマスターです。スキル・経験を積み上げた人はコーチとして様々なチームを指導します。スクラムマスター研修などの講師もその先にあるキャリアです。ユーザー企業・SIer・フリーのどこでも積めるキャリアです。

注意点として、スクラムマスターとしてやっていくには、組織がスクラムマスターという役割を理解している必要があります。従来のプロジェクトマネージャーと混同して、開発者の上司がスクラムマスターを担当することで、開発者が遠慮し、自由に意見交換ができずにいるケースが散見されます。

ただし、熟練のスクラムマスターがいる場合、組織全体にスクラムマスターとは何かを説明し、仲間を増やしながら、組織に合わせた方法で理解を得ていくことが出来ます。素晴らしいスクラムマスター/アジャイルコーチの方々を見ていると、必要なのは観察力、また組織全体やチームに対する少しづつ変化を促せる根気と、失敗を恐れず軌道修正していく勇気だと思います。

3. プロダクトを世に出すプロダクトオーナー

プロダクトビジョンを考え、チームメンバーに説明し、開発内容に責任を持つ人です。ビジネスセンスとプロダクトへの情熱があり、技術にも理解があると鬼に金棒、社内のステークホルダーに話を通せる一定の政治力が必要です。

主にユーザー企業で積むキャリアです。エンジニアの転職先として、これまでではあまり一般的ではなかったかもしれません、視野に入れてみてはいかがでしょうか。エンジニア出身のプロダクトオーナー2名とお会いしたことがあります、開発にもプロダクトにも詳しく、組織内でも頼りにされ、活躍されていました。

4. アジャイルのFWを使って社内のDX推進を行う変革者/コンサルタント

アジャイルは皆さんもご存じの通り、ソフトウェア開発だけではなく、社内変革にも使われ始めています。

人事・予算・開発方法を従来とは違ったやり方で進めていくために、従来のやり方の把握・分析、社内の根回し、各種企画、目玉となるプロダクト開発の立ち上げなどが必要です。コンサルとして他社の変革をサポートするという道もあります。別のロールで一定のキャリアを積んでから、ユーザー企業、SIerなどで変革を担当します。私が目指しているのもこのキャリアです。

5. アジャイルをそれぞれの職種で生かしてキャリアアップする

最後はアジャイルの考え方、工夫を取り入れてそれぞれの職種で生かす道です。人事・育成で、WF開発のプロマネとして、営業として、事務員として……アジャイルは様々な道で生かせます。

まず小さく試し結果を見て改善していく、立場が違う関係者を巻き込みチームとして成果を出していくなど、アジャイルの考え方を取り入れることで、これまでとは違った手応えを感じて仕事をしていくはずです。

20.4 最後に

アジャイルを勉強した後のキャリアの5つのロールモデル、いかがでしたでしょうか。私が考えたもの以外にも、おそらく色んな道があるのではと思います。ぜひ皆さんも自分が目指したいロールモデルについて考えてみてください。



佐竹 朱衣子

株式会社三菱UFJ銀行システム企画部所属（系列IT会社である三菱UFJインフォメーションテクノロジー株式会社より出向）。行内・グループ各社に向けてのアジャイル研修の講師やアジャイル案件への支援などを担当。CSM(Certified Scrum Master)を保有。

第 21 章

intro

イントロ、アジャイルとはを書く

第 22 章

Agile を試した

ほげげ

あとがき

この本を手にとっていただきありがとうございます。

2024年5月
編集長 親方@親方 Project 拝

Agile2

0205 年 2 月 1 日 初版第 1 刷 発行

編 集 親方 Project

発行所 親方 Project

印刷所 株式会社?

(C) 2024 親方 Project