

# スクラム概論

---

第1.1版

2018年08月02日



この作品は [クリエイティブ・コモンズ 表示 - 継承 4.0 国際 ライセンス](https://creativecommons.org/licenses/by-sa/4.0/) の下に提供されています。

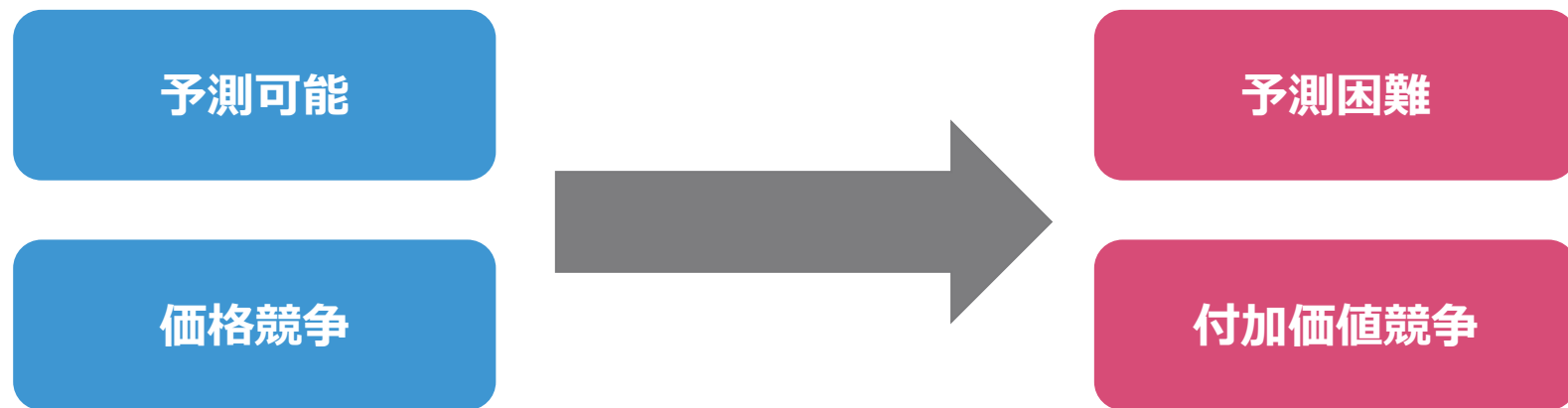
スクラム概論©2018 TIS INC. クリエイティブ・コモンズ・ライセンス（表示-継承 4.0 国際）

# なぜスクラムが求められているのか

---

# 企業を取り巻く環境の変化

出せば売れる時代は“どれだけ投資してどれだけ作れるか”というシンプルな構造だった。  
少々の欠陥があっても売れるので問題なかった。



何が売れるか分からない、**不確実なマーケット**。  
“どれだけ投資してもどれだけのリターンが得られるか”分からない。  
ビジネスモデルの賞味期限が短くなってきている。

マーケットの**激しい変化**に**対応**していかなければならない

# スクラムの定義

---

複雑で変化の激しい問題に対応するためのフレームワークであり、  
可能な限り価値の高いプロダクトを生産的かつ創造的に届けるためのものである。

## 特徴

✓ 軽量



19個の役割とルールしかない。  
PMBOK V6には49個のプロセスが存在する。

✓ 理解が容易

✓ 習得は困難



スクラムは経験主義。  
実際の経験と既知に基づく判断によって習得していく。

スクラムのルールについては、以下のスクラムガイドに記載されている。

<https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Japanese.pdf>

# アジャイル・スクラム・ウォーターフォール

---

# アジャイルとは？

---

Agile – 【形】機敏な、敏捷な

名詞ではなく、**形容詞**。状態を表す。

“ *Don't do agile, be agile* ”

アジャイル開発を行うのが目的ではなく、**アジャイルな状態にする**ことが重要。

その核となる考え方や原則がまとめられたものに、“アジャイルソフトウェア開発宣言”と“アジャイル宣言の背後にある原則”がある。

# アジャイルソフトウェア開発宣言

2001年、当時軽量ソフトウェア開発手法と呼ばれていた分野において著名な17名が集まり、それぞれが重視していた部分を統合し、文書にまとめたもの。

日本で特に知られているのは、以下の6名。

- **Kent Beck** (XPの考案者、JUnitの生みの親)
- **Martin Fowler** (PoEAAの著者。マイクロサービスという言葉の生みの親)
- **Dave Thomas** (達人プログラマーなどの著者。ボブおじさん)
- **Robert C. Martin** (Clean Code, Clean Coderなどの著者)
- **Ken Schwaber** (スクラムの生みの親)
- **Jeff Sutherland** (スクラムの生みの親)

# 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。

<http://agilemanifesto.org/iso/ja/manifesto.html>



# アジャイル宣言の背後にある原則

私たちは以下の原則に従う:

**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。

**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。**

**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。

ビジネス側の人と開発者は、プロジェクトを通して  
日々一緒に働かなければなりません。

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。

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**をすることです。

# アジャイル宣言の背後にある原則(続き)

**動くソフトウェア**こそが**進捗**の最も重要な尺度です。

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きるようにしなければなりません。

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。

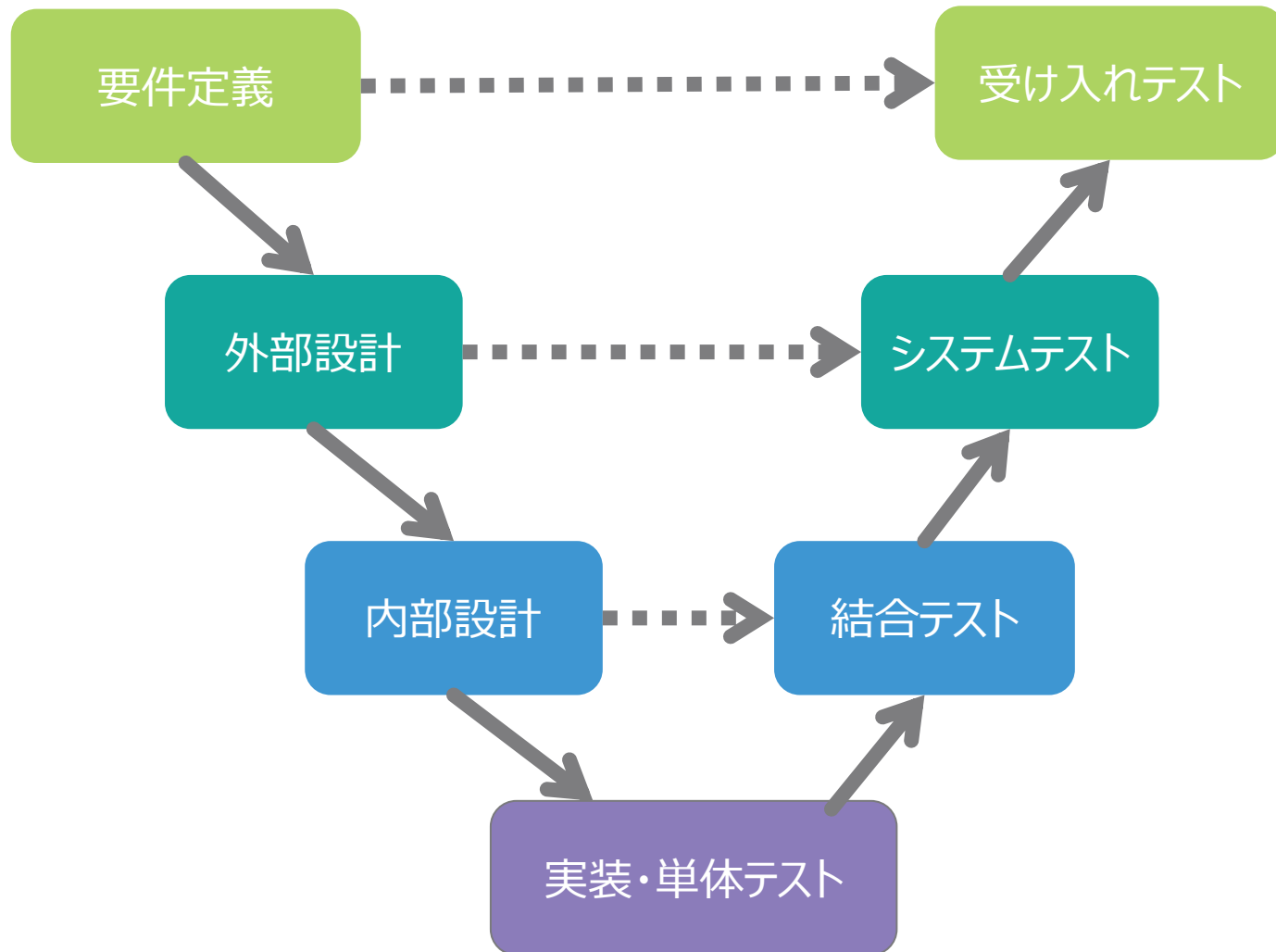
**シンプルさ**（ムダなく作れる量を最大限にすること）が**本質**です。

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。

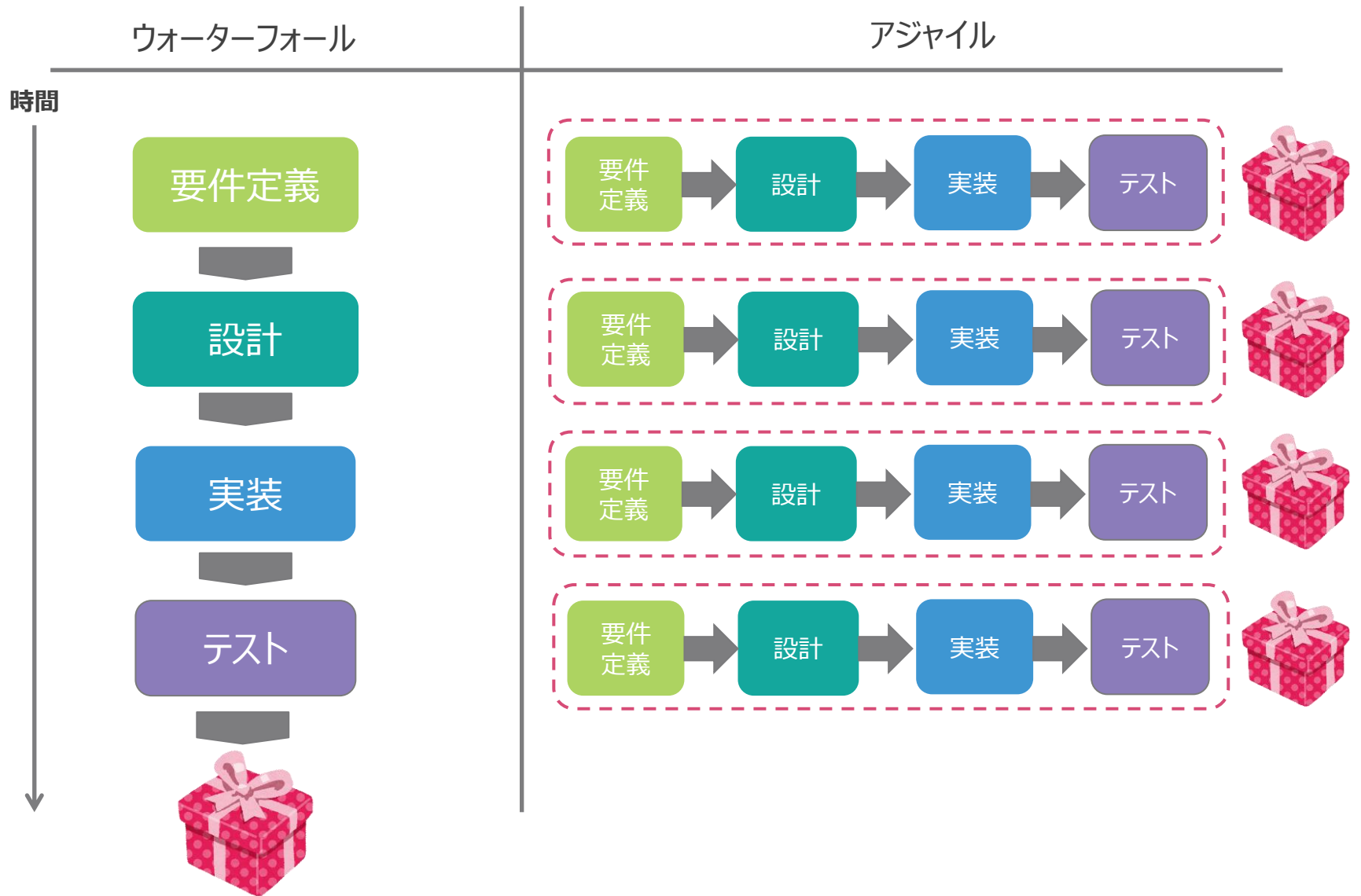
チームがもっと**効率を高める**ことができるかを**定期的**に**振り返り**、  
それに基づいて**自分たちのやり方を最適**に調整します。

# ウォーターフォールの進め方

## おなじみのV字モデル



# アジャイルとウォーターフォールのプロセスの違い



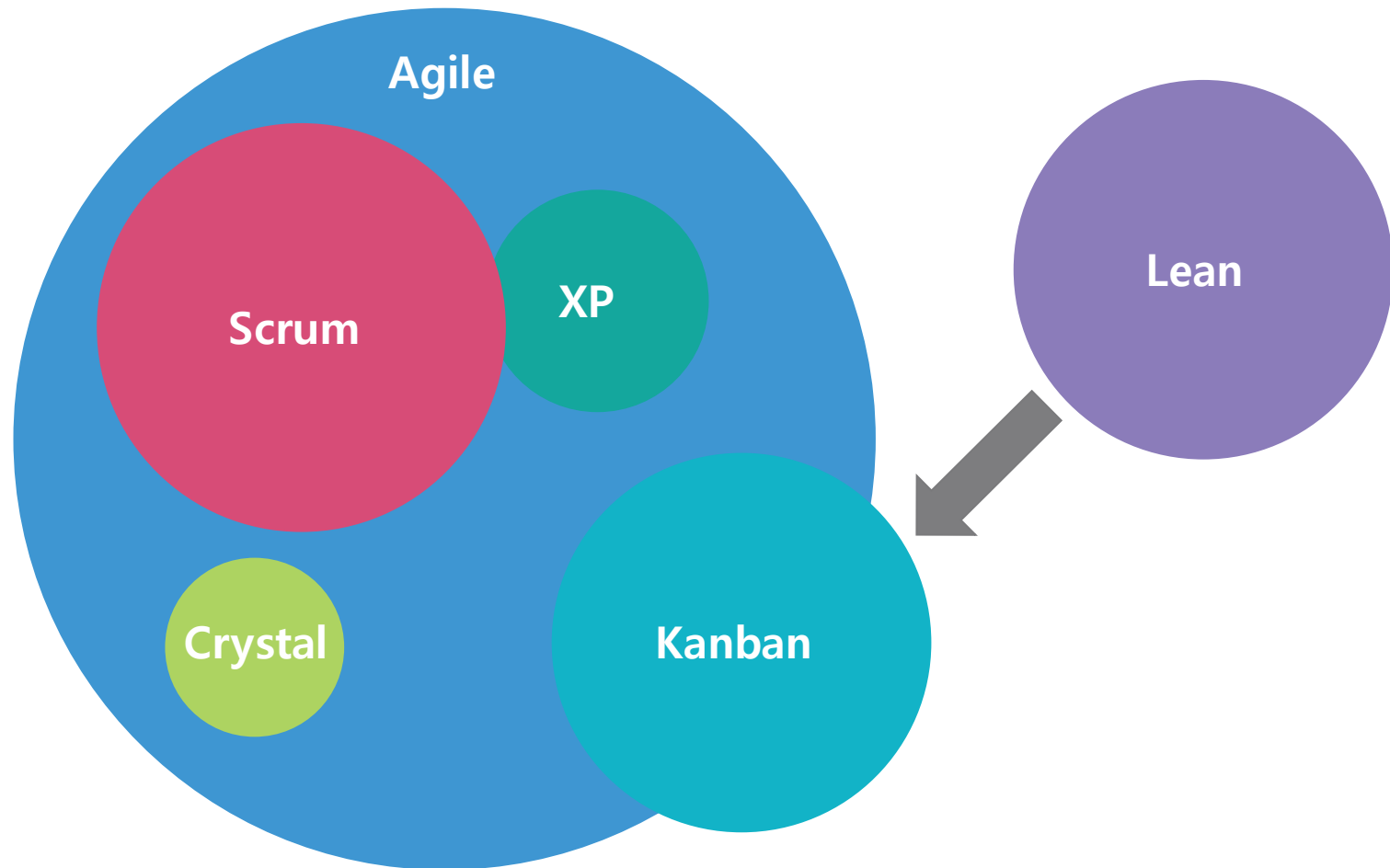
# アジャイルとウォーターフォールのプロセスの違い

プロセス	ウォーターフォール	アジャイル
成功の測定	計画通りに実行出来たか。 QCDによる測定。	変化への対応。 顧客満足。 競争力向上。
マネジメント	指揮命令型 トップダウン	サーバントリーダーシップ フラット
計画	範囲固定で工数を見積る。	期日固定で範囲を見積もる。
要求と設計	最初に要求を洗い出す。 要求のすべてを設計する。	継続的に要求を受け入れる。 必要なタイミングで設計を行う。
実装	全機能を同時に開発。	優先順位の高い機能から開発。
テストと品質保証	終盤に実施。	早期から継続的にテストを実施。

# アジャイルとスクラムの関係

スクラムはアジャイルに包含される。

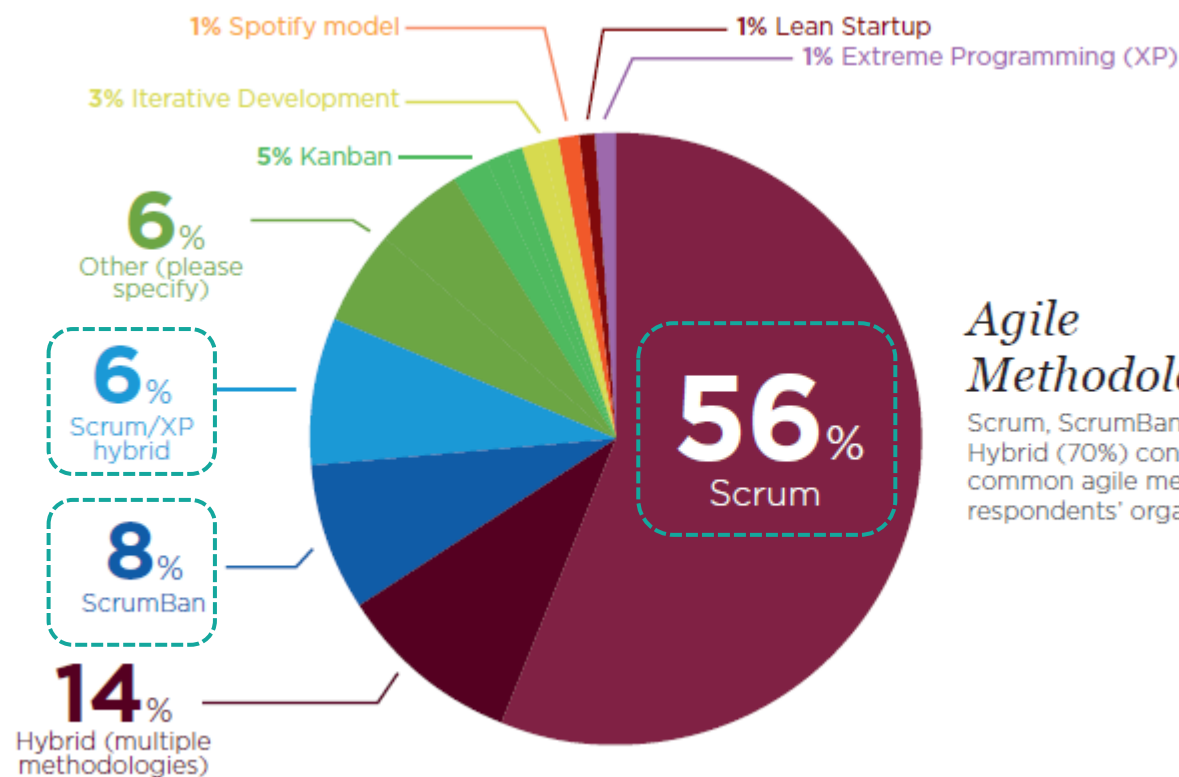
他にもXPやLeanの流れを汲むKanbanなどがアジャイルに含まれる。





# アジャイルの中でのスクラムの採用率

ハイブリッドを合わせると、**70%**がスクラムを採用



## Agile Methodologies Used

Scrum, ScrumBan and Scrum/XP Hybrid (70%) continue to be the most common agile methodologies used by respondents' organizations.

# スクラムとは

---



# スクラムの起源

スクラム自体は**1995年**にJeff SutherlandとKen Schwaberによって発表されているが、スクラムの原点となっている**野中郁次郎**、**竹内弘高**の研究論文「The New New Product Development Game」は**1986年**に発表されている。

論文の中で、柔軟で自由度の高い日本発の開発手法をラグビーの**スクラム**に喩えて「**Scrum**（スクラム）」として紹介した。

実は、アジャイルよりスクラムの方が歴史は長い



Luke Burgess introduces the ball into the scrum.

[https://ja.wikipedia.org/wiki/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:ST\\_vs\\_Gloucester\\_-\\_Match\\_-\\_23.JPG](https://ja.wikipedia.org/wiki/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:ST_vs_Gloucester_-_Match_-_23.JPG)

# スクラムの定義(再掲)

---

複雑で変化の激しい問題に対応するためのフレームワークであり、  
可能な限り価値の高いプロダクトを生産的かつ創造的に届けるためのものである。

## 特徴

✓ 軽量



**19**個の役割とルールしかない。  
PMBOK V6には**49**個のプロセスが存在する。

✓ 理解が容易

✓ 習得は困難



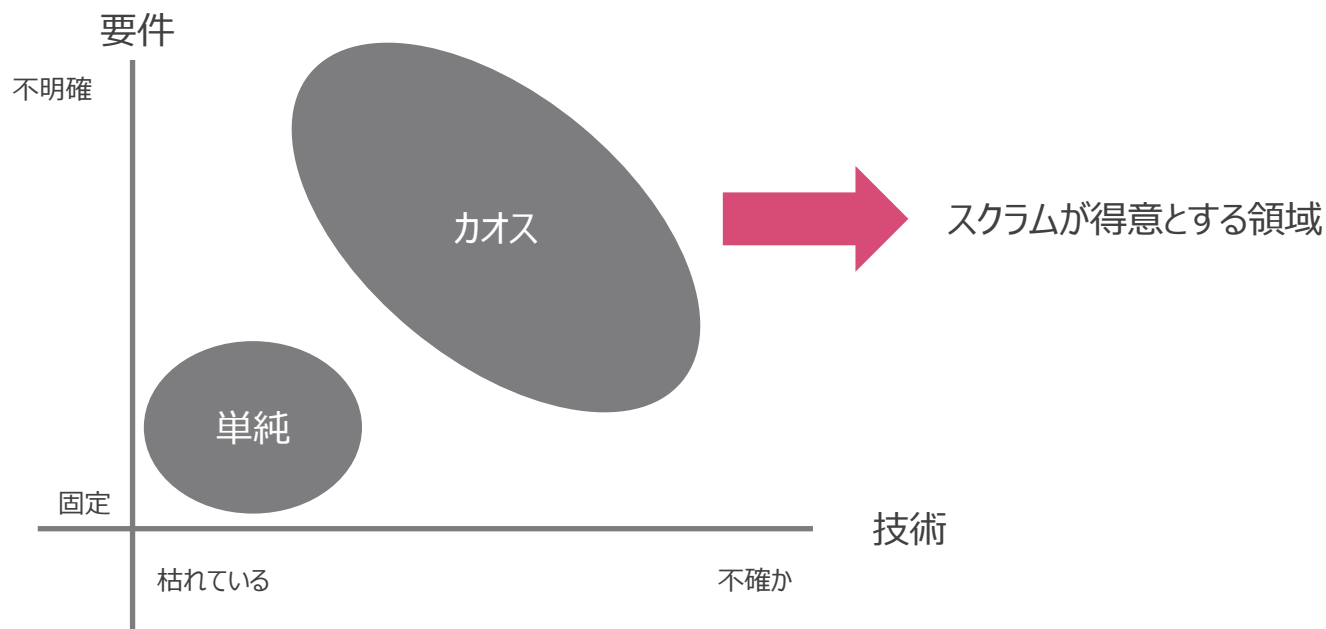
スクラムは経験主義。  
実際の経験と既知に基づく判断によって習得していく。

スクラムのルールについては、以下のスクラムガイドに記載されている。

<https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Japanese.pdf>

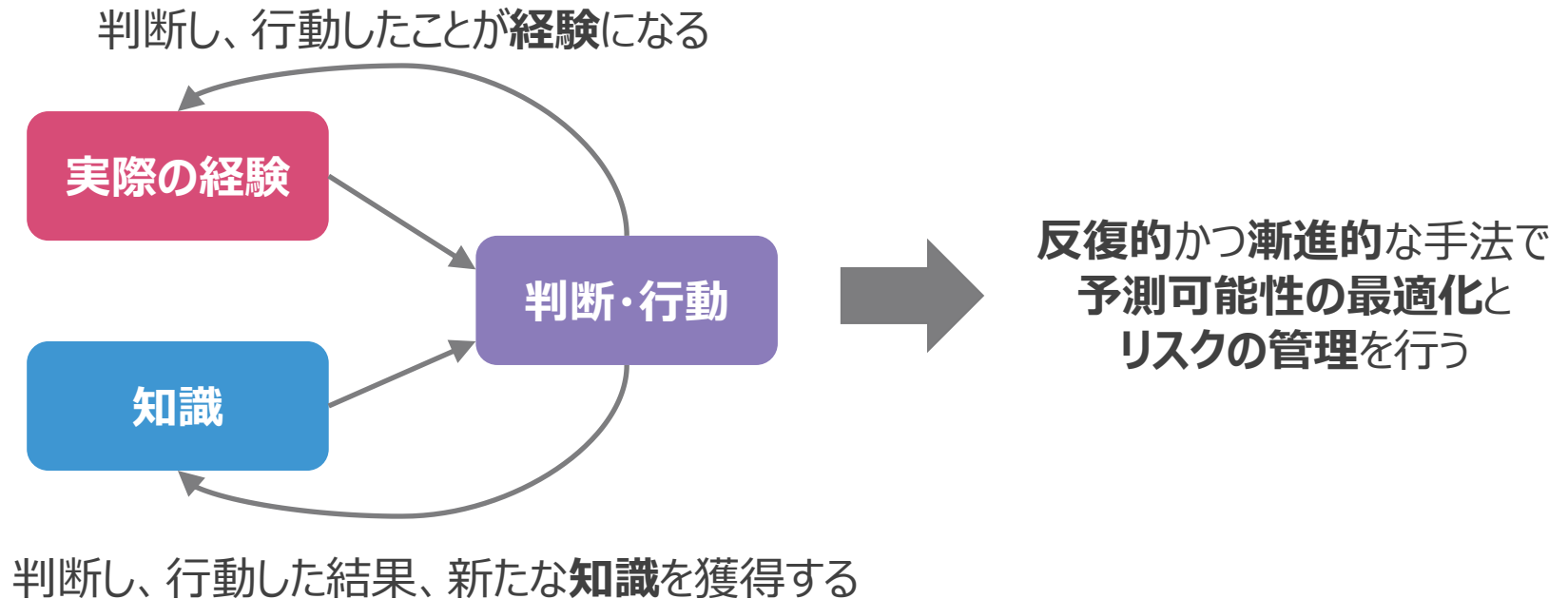
# スクラムに向かないプロジェクト

- プロジェクトの状態が**単純**
- **決まりきったモノ**を作る
- チームの**生存期間が短い**
  - 3ヶ月より短いと技術やプロセスに習熟しない



# スクラムの理論

スクラムは、経験的プロセス制御の理論(**経験主義**)を基本にしている。



# スクラムを支える3本柱

---

経験的プロセス制御の実現は以下の3つによって支えられている。

## 透明性 (Transparency)

結果責任を持つ人に対して**見える化**されていること。  
標準化され、見ている人が**共通理解**を持てること。

## 検査 (Inspect)

作成物や進捗を頻繁に検査し、**変化を検知**する。  
検査を頻繁にやりすぎて作業の妨げになってはいけない。

## 適応 (Adapt)

プロセスに不備がある場合、その構成要素を調整する。  
プロセスの調整を出来る限り早く行い、逸脱を防ぐ。

スクラムのイベント、作成物、ロールには必ず何れかが当てはまる。

# スクラムの5つの価値基準

---

5つの価値基準を取り入れ、実践することで透明性・検査・適応が実現可能となる。

## 確約 (Commitment)

**個人**はスクラムチームの**ゴールの達成**を**確約**しなければならない。  
無理をして絶対に終わらせるということではない。

## 勇気 (Courage)

**スクラムチームのメンバー**は、**正しいことをする勇気**を持ち、  
困難な問題に取り組まなければならない。

## 集中 (Focus)

**スクラムチーム全員**が**スプリントの作業とゴール**に**集中**しなければならない。

## 公開 (Openness)

**スクラムチームとステークホルダー**は、**仕事や課題**と、その**遂行の様子**を  
**公開**することに合意しなければならない。

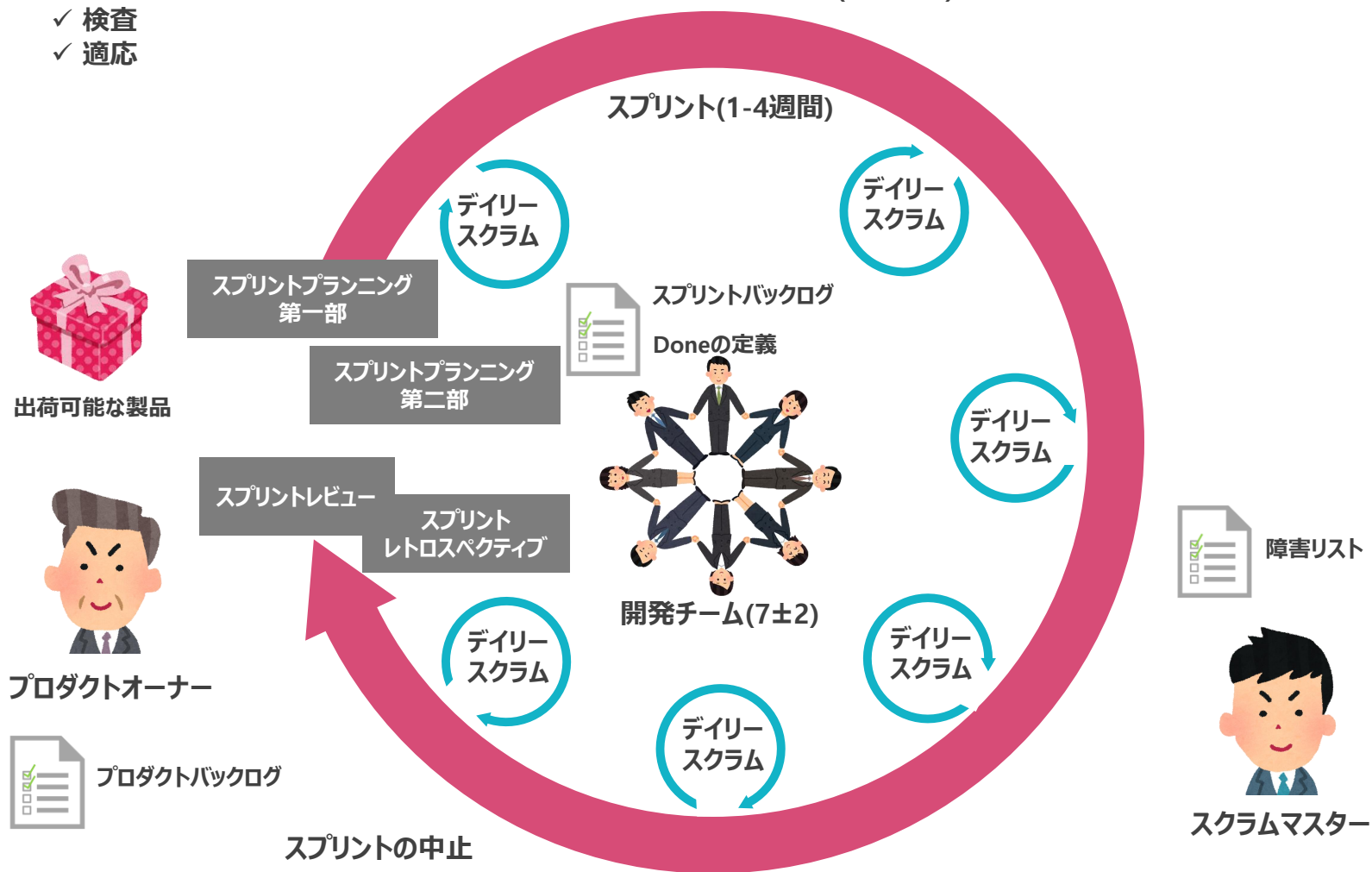
## 尊敬 (Respect)

**スクラムチームのメンバー**は、**お互いを能力のある独立した個人**として  
**尊敬**しなければならない。

# スクラムにおける19のルール

- ✓ 透明性
- ✓ 検査
- ✓ 適応

プロダクトバックログリファインメント(5~10%)



# スクラムにおける3つのロール

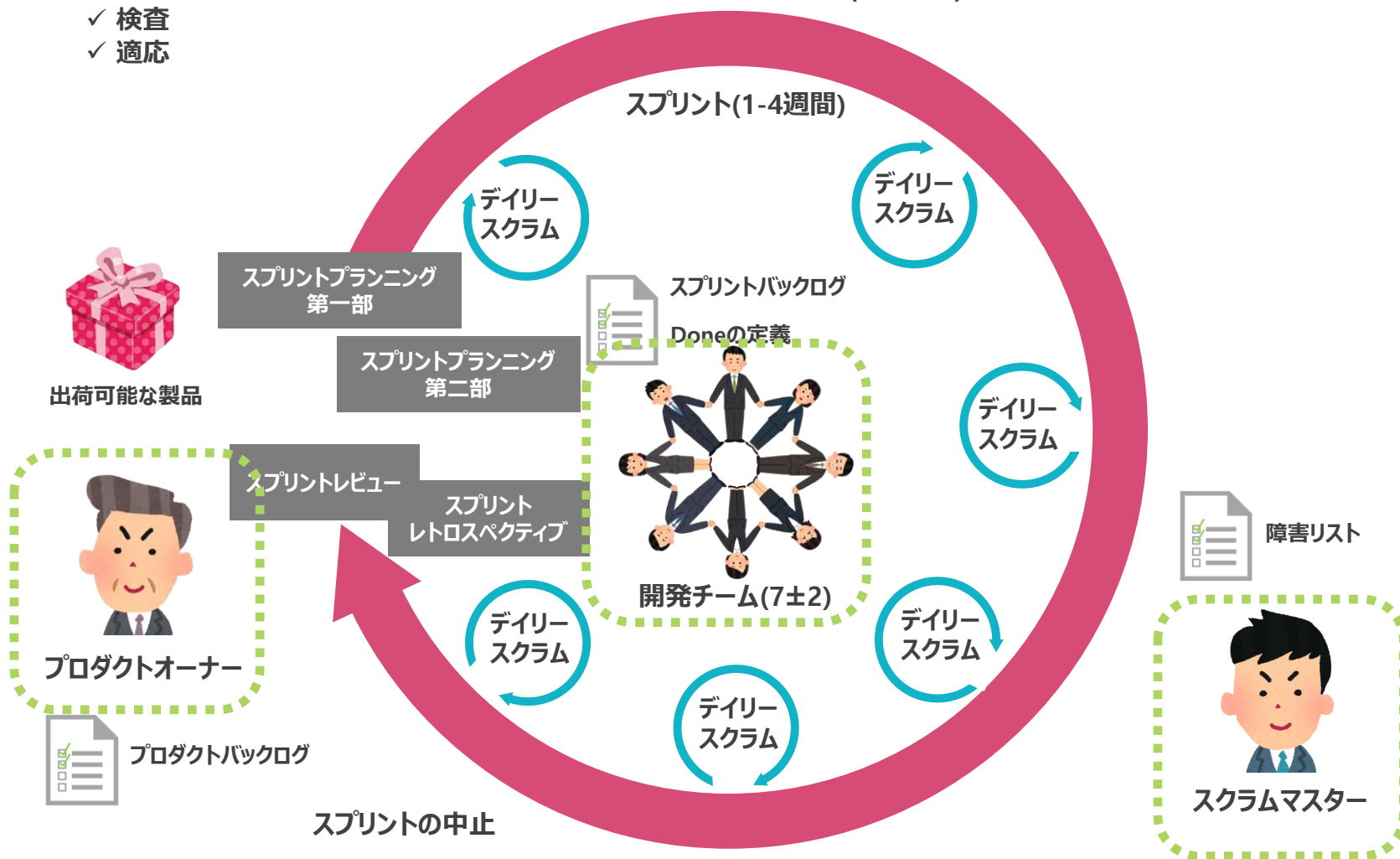
---



# スクラムにおける3つのロール

- ✓ 透明性
- ✓ 検査
- ✓ 適応

プロダクトバックログリファインメント(5~10%)



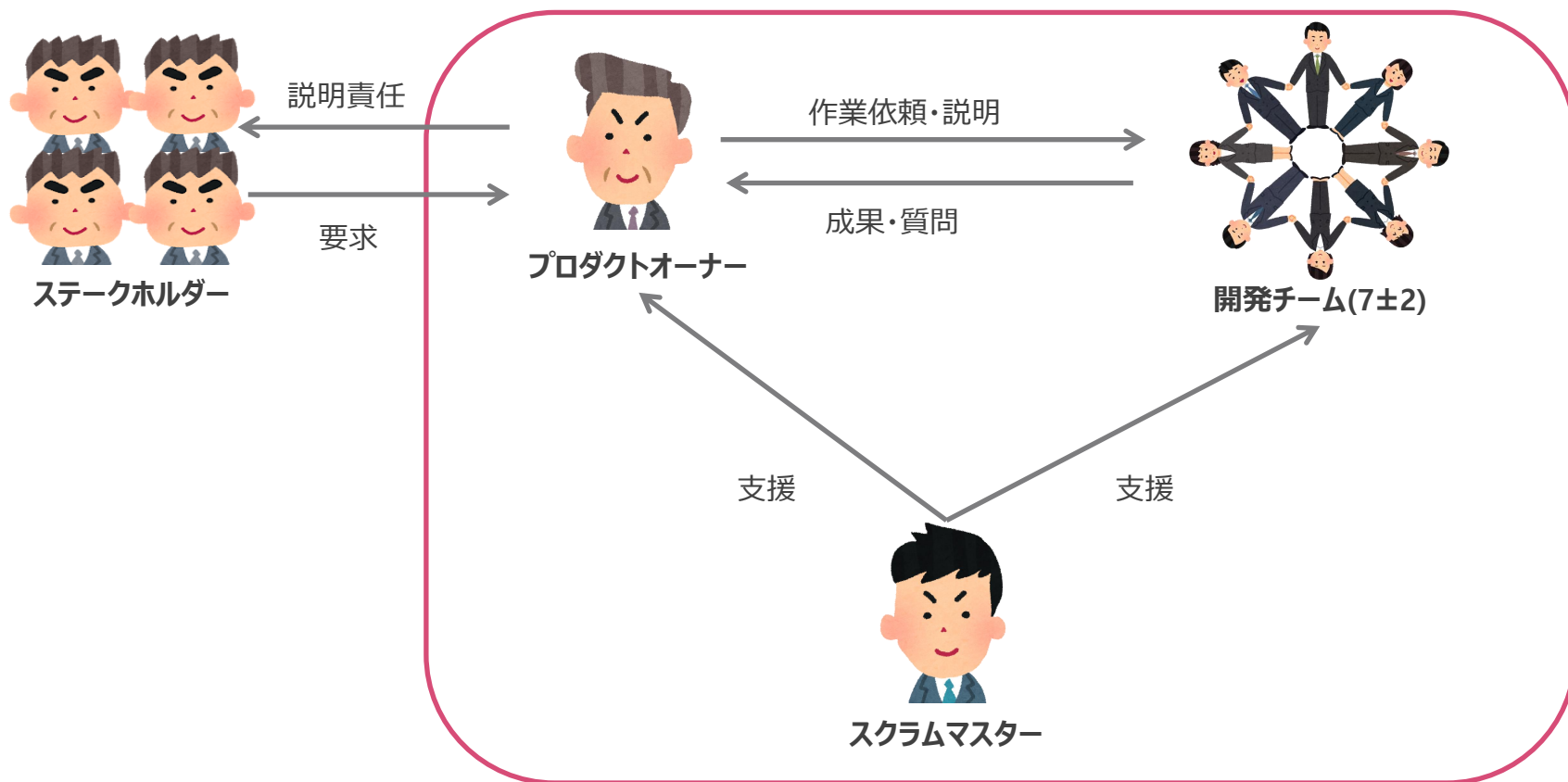
# スクラムチーム

プロダクトオーナー、開発チーム、スクラムマスターで構成される。

**自己組織化**されており、**機能横断的**。

ゴールを達成するための**最善策を自分たちで選択する**。

## スクラムチーム



# プロダクトオーナー(PO)

## 求められる能力

- コスト感覚
- ネゴシエーション力
- 説明力
- 決断力
- 一貫性
- 戦略性

## 役割

- 開発チームの作業とプロダクトの価値の最大化に責任を持つ
- リリース日、リリース内容を決める
- プロダクトバックログの管理に責任を持つ**1人の人間**
- プロダクトバックログの優先順位の決定(委員会があっても構わないが、決定はPOの責任)
- 開発チームへの作業依頼(PO以外が開発チームに作業依頼をしてはいけない)
- 作業結果の受入・拒否

## 仕事

- プロダクトバックログアイテム(PBI)を明確に表現する
- ゴールとミッションを達成できるようにPBIを並べ替える
- 開発チームが行う作業の価値を最適化する
- PBIを全員に見える化・透明化・明確化し、スクラムチームが次に行う作業を示す
- 必要とされるレベルでPBIを開発チームに理解してもらう



# 開発チーム

## 求められる能力

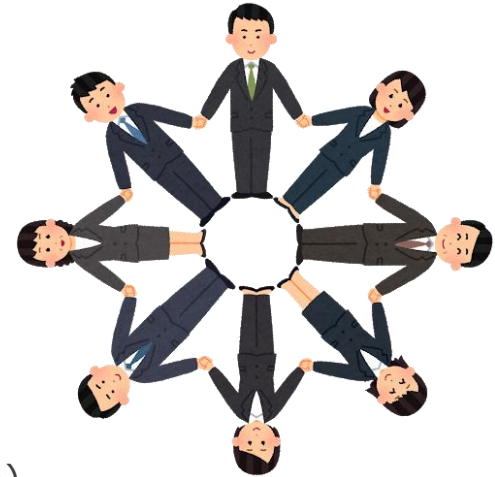
- 開発力
- 自己組織化
- 見積力
- **職能横断的**スキル(全員揃えばプロダクトを作れる)

## 役割

- リリース可能なモノを作成する
- 自分たちの作業の管理(**1番良いやり方を自分たちで考え、決める**)
- **生産性を向上するために努力する責任**

## 仕事

- スプリントプランニングで約束したことを実現する
- 生産性向上のための行動を常に考え、行う



# スクラムマスター

---

## 求められる能力

- サーバントリーダーシップ
- ティーチング力
- ファシリテーション力
- コーチング力
- スクラム以外のプロセスの理解
- 集団心理の理解
- 事実(数字)を示す

## 役割

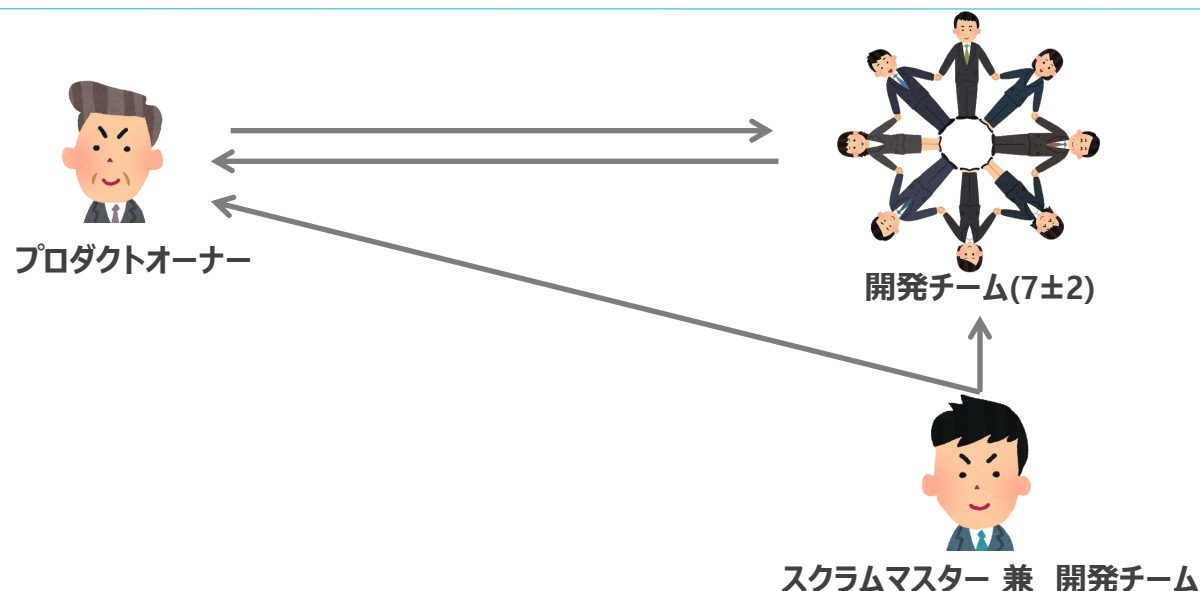
- **スクラムの理解と成立**に責任を持つ
- プロダクトオーナーの支援
- 開発チームの支援

## 仕事

- スクラムの説明を行い、理解してもらう
- 効果的なプロダクトバックログの管理方法の検討
- (必要に応じて) イベントのファシリテート
- 開発チームの進捗を阻害する要因の排除



# スクラムチームアンチパターン – 兼任スクラムマスター



## 課題

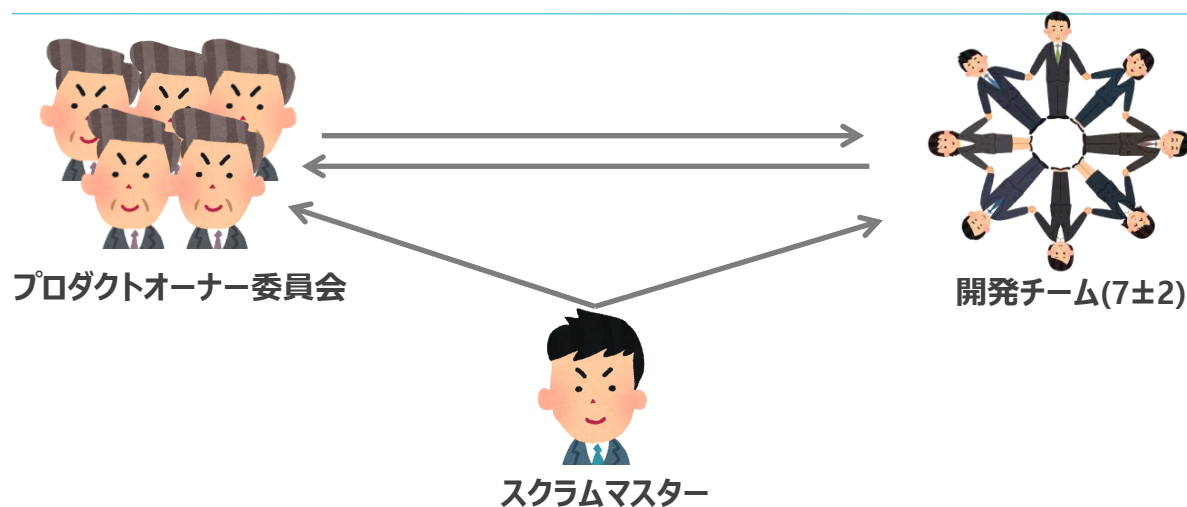
スクラムマスターとして開発チームに対して厳しく接する場面もあるが、その際にスクラムマスターとしての発言なのか開発チームの一員としての発言なのか、受け取る側が混乱しやすい。

特に、従来の開発でチームリーダーを担当していたような人は技術的にも優れている場合が多いため、アーキテクチャの検討や設計・開発とスクラムマスターとしての役割を両立することになり負担が大きい。

## 解決策

スクラムマスターの経験がある場合、チーム内でスクラムマスターや技術的に頼れる人を育て、手放していく。経験が無いのであれば開発チームからスクラムマスターを選び、自身は開発者に専念する。

# スクラムチームアンチパターン – PO委員会



## 課題

プロダクトオーナーが複数人いるため、意見が統一されず開発チームが混乱する。  
プロダクトオーナー委員会内で意見が衝突する場合があるため、判断スピードが遅くなる。  
開発チームが仕様を問い合わせる際に誰に連絡を取れば良いかわからない。

## 解決策

委員会があっても良いが、必ずプロダクトオーナーを1人決める。  
スクラムマスターはプロダクトオーナー以外からの作業依頼はブロックする。

# スクラムチームアンチパターン – 意欲に満ちたステークホルダー



## 課題

積極的に協力しようと、開発チームにアドバイスや情報を提供してくれるがプロダクトオーナーとすり合わせられていない。また、プロダクトバックログに反映するための調査作業などを直接開発チームに依頼してしまう。

プロダクトオーナーが把握していない作業や情報が増え、ベロシティの低下や、本来優先すべきタスクが後回しにされる事態を招く。

## 解決策

スクラムマスターはステークホルダーからの直接の依頼をブロックし、プロダクトオーナーを必ず通すように説明する。プロダクトオーナーはステークホルダーに対して情報収集と説明を行う。



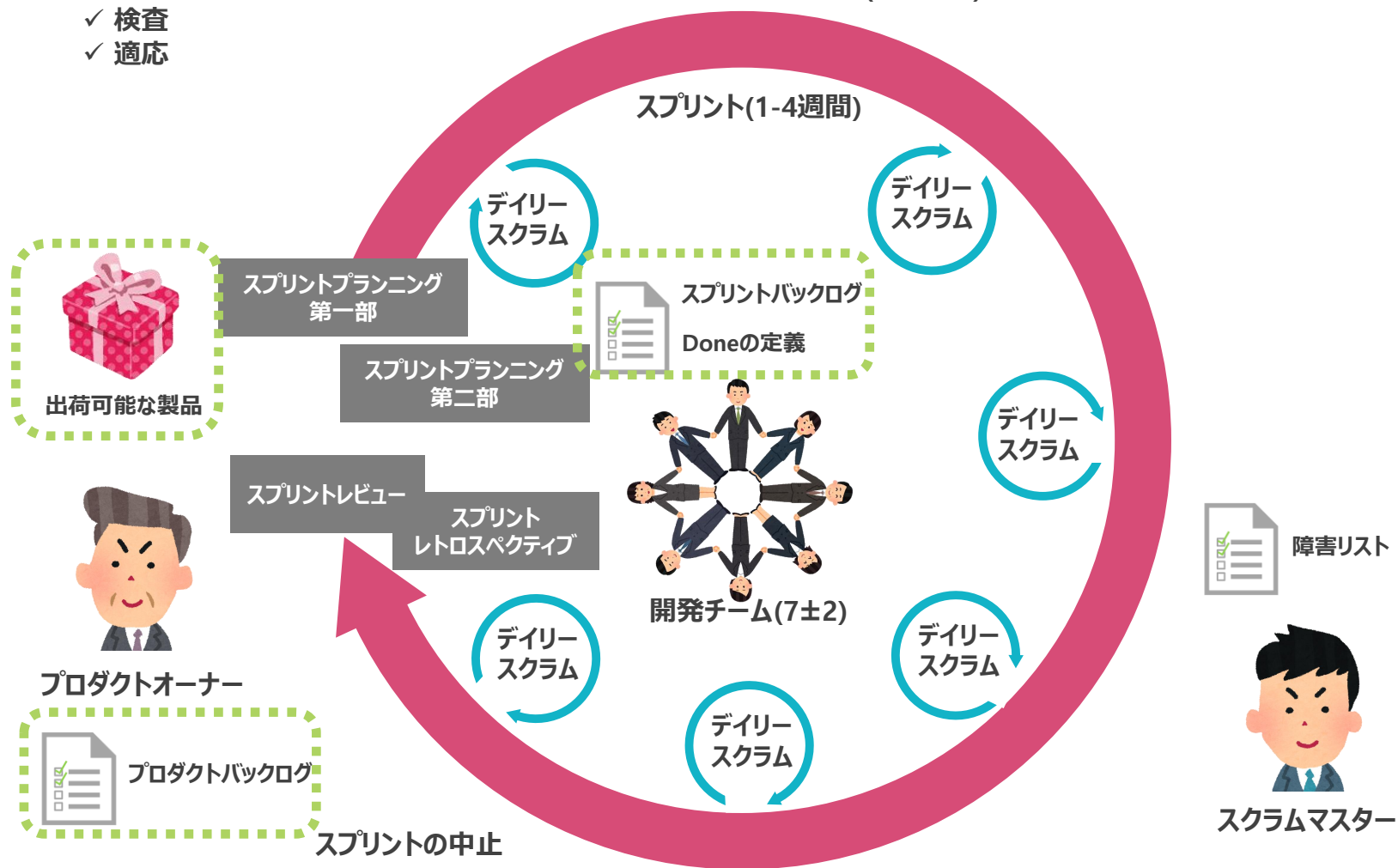
# スクラムの3つの作成物

---

# スクラムの3つの作成物

- ✓ 透明性
- ✓ 検査
- ✓ 適応

プロダクトバックログリファインメント(5~10%)

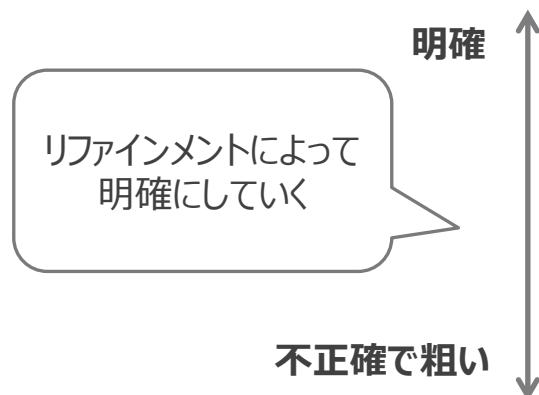


# プロダクトバックログ

プロダクトに必要なものが全て**優先順位**で並んだ一覧。  
プロダクトに対する**変更要求の唯一の情報源**となる。  
プロダクトバックログの1要素をプロダクトバックログアイテム(PBI)と呼ぶ。  
プロダクトバックログはビジネス要求、市場の状態、技術の変化などにより、**常に変化し続ける**。

PBIに内容の詳細や、見積り、並び順の変更などを加えることを  
**プロダクトバックログリファインメント**と呼ぶ。  
リファインメントのタイミングはスクラムチームが決定する。  
一般的に開発チームの作業の10%以下にすることが多い。

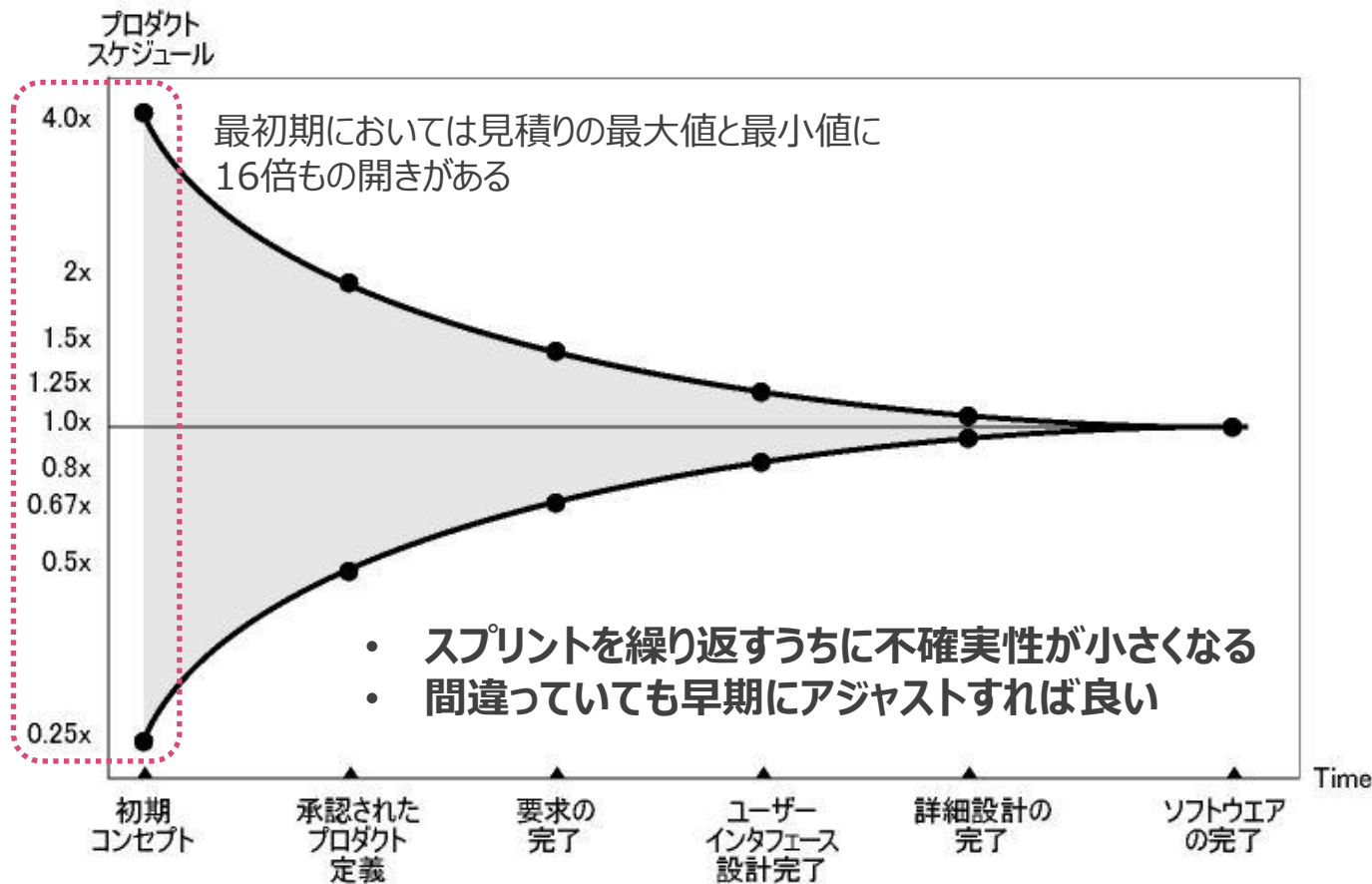
見積りは**相対見積り**で行う。値はフィボナッチ数列やS,M,Lなど、なんでも良い。



優先順位	ストーリー	見積
1	AとしてXXが出来る。	2
2	BとしてYYを一覧形式で参照できる。	3
3	C処理の性能改善	5
...	...	...
100	Dとしてレポートを作成できる	8

# 不確実性コーン

プロジェクトが進行するにつれて見積りのバラツキがどのように推移していくのかを表している。

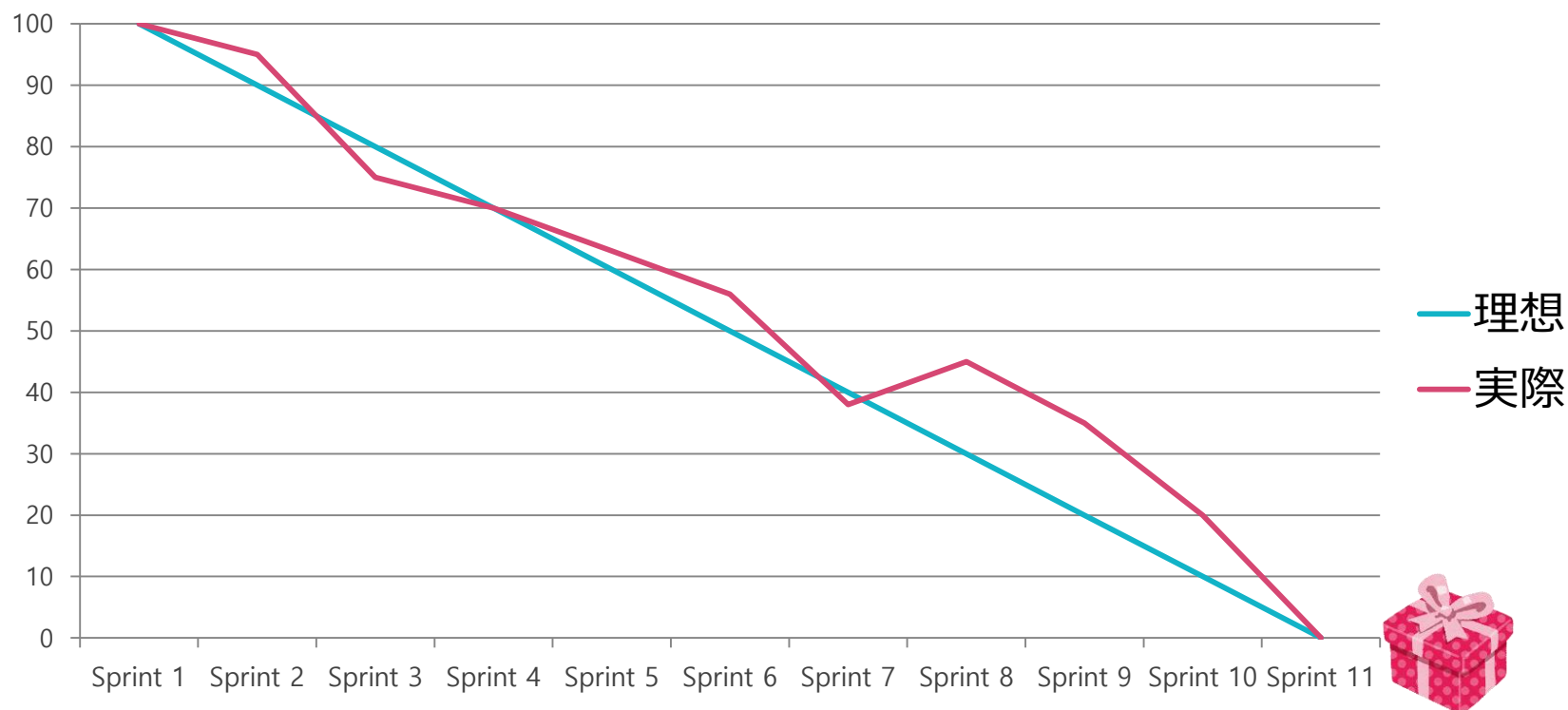


<http://itpro.nikkeibp.co.jp/article/COLUMN/20131001/508039/>

# ゴールへの進捗の確認

リリースバーンダウンチャートなどを用いて、希望している内容が希望しているタイミングでリリースできるかを追跡、確認する。スプリントレビュー時に確認するとプロダクトオーナーにスコープの調整の材料を与えることができる。

リリースに必要なポイント数とタイミングが分かれば、1 スプリント実施するだけで遅れの有無がわかる。



# スプリントバックログ

スプリントで選択したプロダクトバックログアイテムと、それらを出荷可能な製品として届けるための計画を合わせたもの。

開発チームが**スプリントゴールを達成するのに必要な作業の一覧**。

十分に詳細化されており、スプリント中は変更される可能性がある。

スプリントバックログを変更できるのは開発チームだけ。

見積りは**理想時間**で行う。

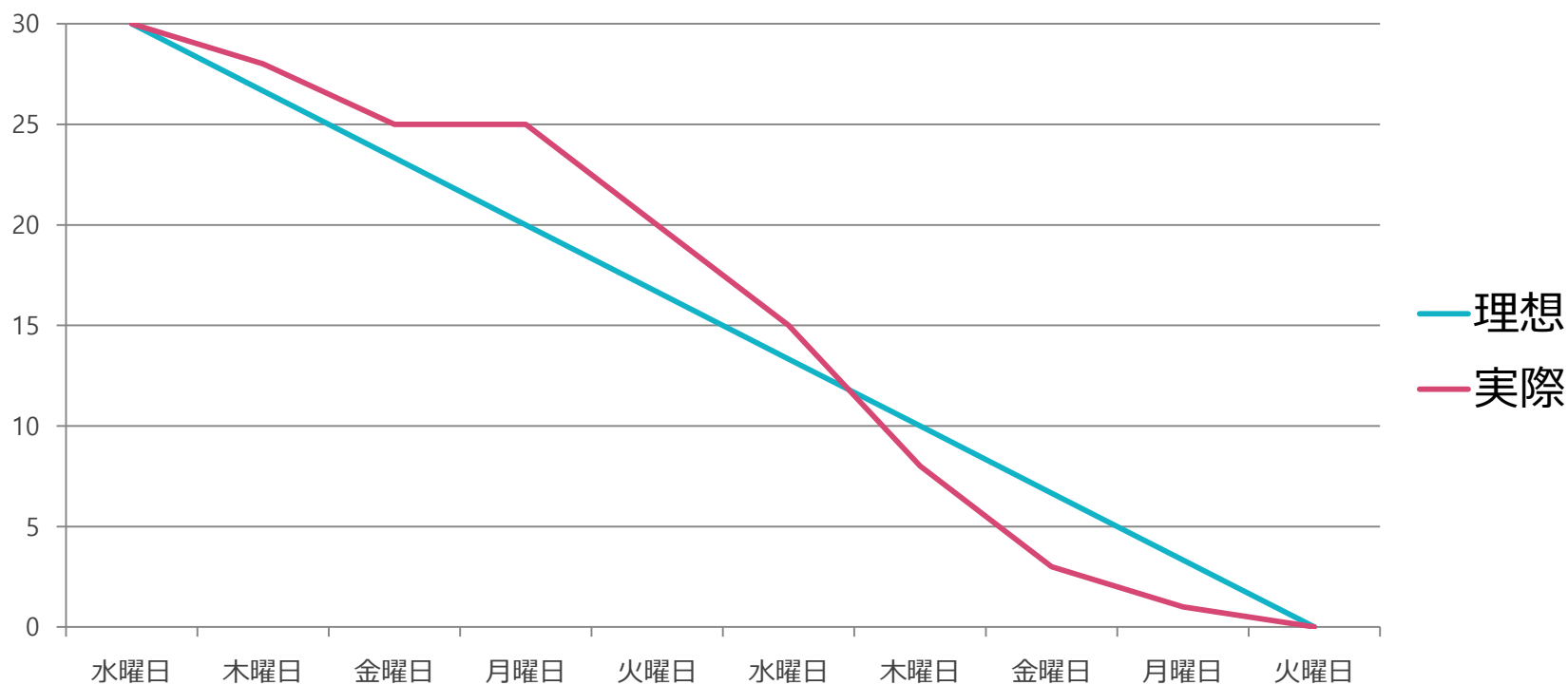
タスクの粒度は小さい方が見積り精度が上がるため、小さい方が良い。

大きくても1日のうち開発作業にあてられる時間程度を上限とすること。

ストーリー	タスク	見積
AとしてXXが出来る。	UIのコーディング	3.0h
	データモデル設計、変更、Entityの作成	3.0h
	Actionのコーディング	2.0h
BとしてYYを一覧形式で参照できる。	UIのコーディング	4.0h
	Actionのコーディング	2.0h

# スプリントの進捗の確認

スプリントバックログの残作業量を追跡し、進捗を確認する。  
デイリースクラムでバーンダウンチャートを確認することが多い。



## 出荷可能な製品

これまで作成してきた製品と、今回のスプリントで完成したプロダクトバックログアイテムを合わせたもの。

スプリントの終わりには**出荷可能**な製品が完成していなければならない。

出荷可能とは、スクラムチームの**完成の定義**に合っていることを意味する。

部品だけでは出荷できない。**小さくても使えるもの**を作る





# 作成物の透明性

---

# Done(完成)の定義

出荷可能な製品の「完成」の定義。作業が完了したかどうかの評価に使われる。  
リリースするためにやらなければならないこと。

## Done

毎スプリント完了しているもの

- 開発
- UT
- カバレッジ80%以上
- IT
- リグレッションテスト
- ドキュメント作成
- 静的解析

## Undone

毎スプリント実施できていないもの

- シナリオテスト
- 負荷テスト
- セキュリティテスト
- リリース



**UndoneをDoneにしていくことが重要**

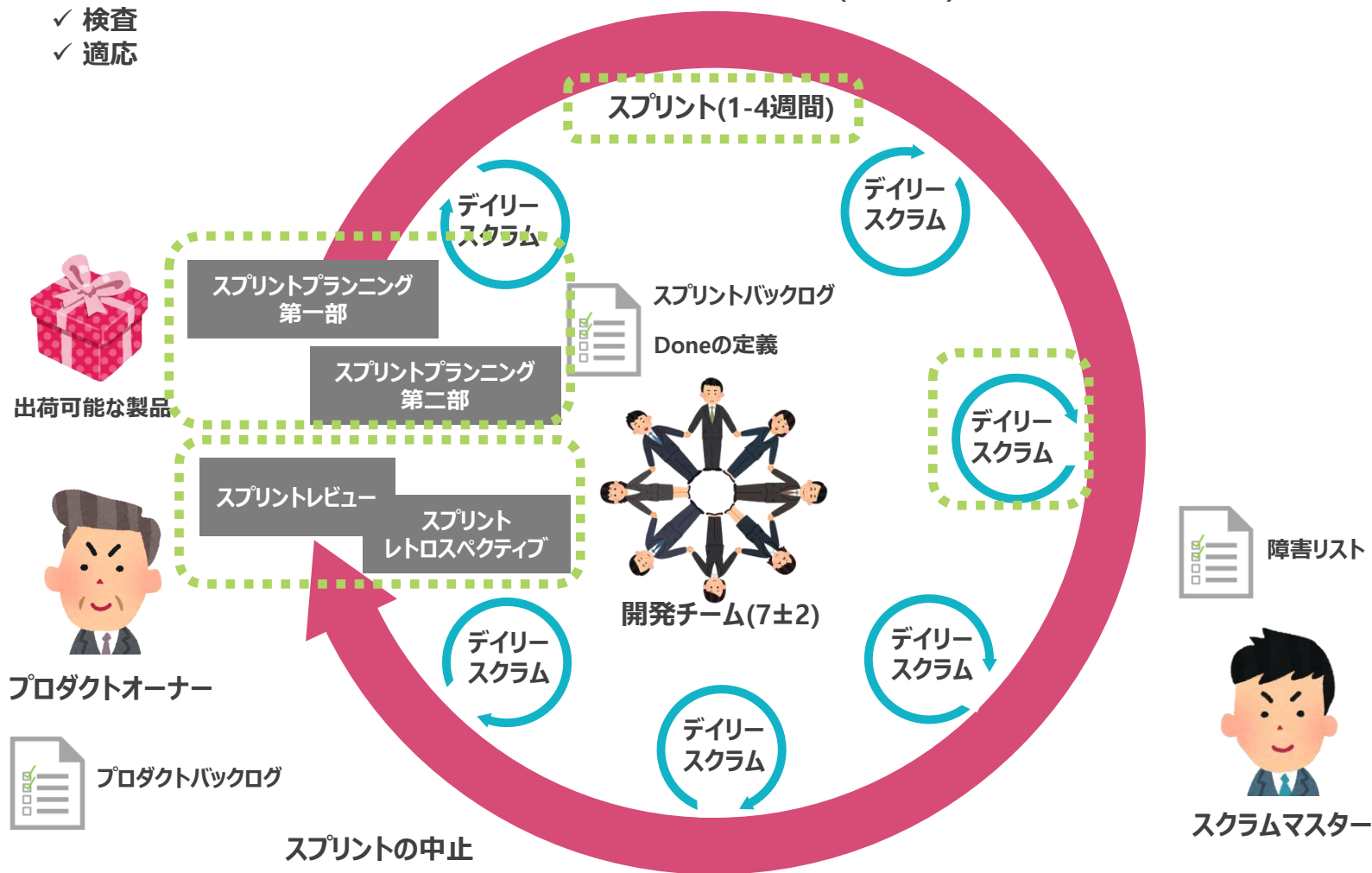
# スクラムにおける5つのイベント

---

# スクラムにおける5つのイベント

- ✓ 透明性
- ✓ 検査
- ✓ 適応

プロダクトバックログリファインメント(5~10%)



# スプリントプランニング

タイムボックス: 4時間 / 2週間

## スプリントプランニング第一部

「スプリントの成果である出荷可能な製品の増分として何を届けることができるか？」という問いに答える。インプットはプロダクトバックログ、最新の製品、開発チームの予想キャパシティと実績。これを元にどのプロダクトバックログアイテムまで選択するのかを決定する。この**アイテム数**については開発チームが責任を持つ。

スプリントで届けるプロダクトバックログを**予想**したあとに、**スプリントゴールを設定**する。スプリントゴールは、**プロダクトバックログを実装することで実現するスプリントの目的**。

優先順位	ストーリー	見積
1	AとしてXXが出来る。	2
2	BとしてYYを一覧形式で参照できる。	3
3	C処理の性能改善	5
...	...	...
100	Dとしてレポートを作成できる	8

スプリントで実施するPBIを選択

# スプリントプランニング

## スプリントプランニング第二部

「出荷可能な製品の増分を届けるために必要な作業をどのように成し遂げるか？」という問に答える。  
第一部で実施すると決めたプロダクトバックログアイテムを、スプリントバックログに落とし込む。

スプリントの最初の数日間分のタスクについては、この段階で作業レベルまで**タスクを分解**する。  
タスクの分解は、必要に応じてスプリント期間中も実施する。

プロダクトオーナーはプロダクトバックログの明確化やトレード・オフを支援する。  
タスクを分解した結果、作業が多すぎたり少なすぎたりする場合はプロダクトオーナーと話し合い、調整する。

開発チームは、どのようにスプリントゴールを達成するかを説明できなければならない。

ストーリー
AとしてXXが出来る。
BとしてYYを一覧形式で参照できる。

具体的なタスクに分解する



タスク	見積
UIのコーディング	3.0h
データモデル設計、変更、Entityの作成	3.0h
Actionのコーディング	2.0h
UIのコーディング	4.0h
Actionのコーディング	2.0h

# スプリント

**タイムボックス:** 1ヶ月以下

出荷可能な製品を作成するための1ヶ月以下のタイムボックスであり、開発作業を行う連続した期間。スプリントはスプリントプランニング、デイリースクラム、開発作業、スプリントレビュー、スプリントレトロスペクティブで構成される。

## 注意

- スプリントゴールに悪影響を及ぼすような変更は加えない
- 品質目標は下げない
- 学習が進むにつれてスコープが明確化され、プロダクトオーナーと開発チームの交渉が必要になる可能性がある

## スプリントの中止

スプリントはタイムボックスの終了前に中止することができるが、**スプリントの中止の権限があるのはプロダクトオーナーだけ。**

中止した場合は、プロダクトバックログの完成したアイテムをレビューする。

未完成のプロダクトバックログアイテムは、再見積りを行ってからプロダクトバックログに戻す。



# デイリースクラム



**タイムボックス:** 15分 / 1日

前回のデイリースクラムから行った作業の検査と、次回のデイリースクラムまでに行う作業の計画を立てる。

この場でスプリントバックログの**作業進捗を検査**する。  
デイリースクラムは**毎回同じ時間、場所**で開催する。

デイリースクラムでは、開発チームのメンバーが以下のことを説明する。

- 開発チームが**スプリントゴールを達成するために、私が昨日やったことは何か？**
- 開発チームが**スプリントゴールを達成するために、私が今日やることは何か？**
- 私や開発チームが**スプリントゴールを達成するときの障害物を目撃したか？**

デイリースクラムで詳細な内容に踏み込みそうな場合は、デイリースクラム終了後に開発チームまたは一部のチームメンバーで集まり、詳細な議論、適応、再計画を行う。





# スプリントレビュー



**タイムボックス:** 2時間 / 2週間

スプリントの終わりに出荷可能な製品の増分の検査と、必要であればプロダクトバックログの適応を行う。スプリントレビューでは、スクラムチームと関係者がスプリントの成果をレビューする。

**進捗確認の場ではなく、フィードバックや更なる協力を引き出すことが目的。**

スプリントレビューには以下が含まれる。

- ✓ 参加者はプロダクトオーナーが招待する
- ✓ **プロダクトオーナー**がPBIの完成したものと完成していないものについて説明する
- ✓ 開発チームはスプリントでうまくいったこと、直面した課題、それをどう解決したかを議論する
- ✓ 開発チームは完成したものをデモして、質問に答える
- ✓ プロダクトオーナーは現在のプロダクトバックログを確認し、完了日を予測する
- ✓ グループ全体で次に何をするか議論し、次のスプリントプランニングのインプットにする
- ✓ プロダクトの次のリリースに対するスケジュール、予算、性能、市場をレビューする



# スプリントレトロスペクティブ



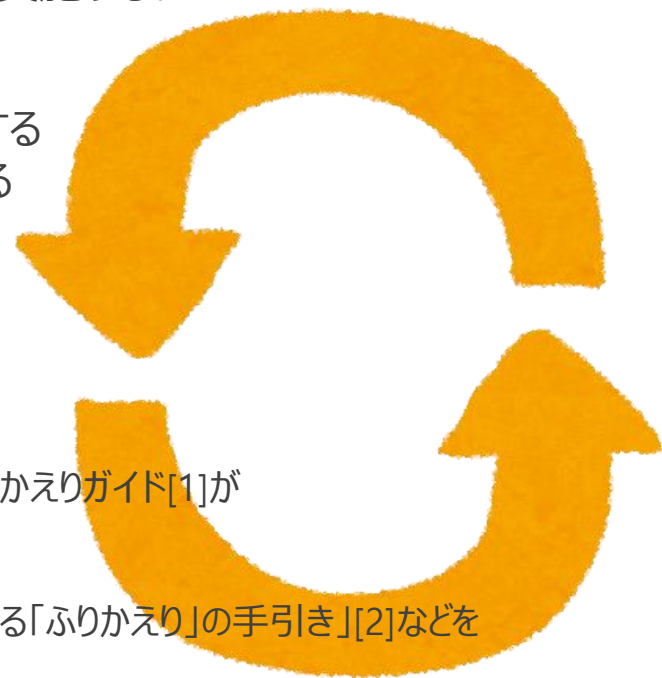
タイムボックス: 1.5時間 / 2週間

**スクラムチーム**の検査と次のスプリントの改善計画を作成する機会。  
スプリントレビューが終わり、次のスプリントプランニングが始まる前に実施する。

- 人・関係・プロセス・ツールの観点から今回のスプリントを検査する
- うまくいった項目や今後の改善が必要な項目を特定・整理する
- 次のスプリントで実施する改善策を特定する
- スクラムチームの作業の改善実施計画を作成する
- 完成の定義を適切に調整する

方法は特に規定されていないが、よくKPTが用いられる。  
KPTの実践的な進め方についてはプロジェクトファシリテーション実践編 ふりかえりガイド[1]が参考になる。





















その他の方法については、「アジャイルレトロスペクティブズ 強いチームを育てる「ふりかえり」の手引き」[2]などを参照。



[1]: <http://objectclub.jp/download/files/pf/RetrospectiveMeetingGuide.pdf>

[2]: <http://shop.ohmsha.co.jp/shopdetail/000000001770/>

# 会議体及び出席者

ミーティング出席者 ガイドライン	スプリント プランニング 第一部	スプリント プランニング 第二部	デイリースクラム	スプリントレビュー	スプリント レトロスペクティブ
<b>プロダクトオーナー</b> 開発チームの作業とプロダクトの 価値の最大化に責任を持つ。 プロダクトバックログの管理に責任 を持つ。		 [1]	 [2]		 [4]
<b>スクラムマスター</b> スクラムの理解と成立に責任を 持つ。			 [3]		
<b>開発チーム</b> プロダクトの開発を行う。 スプリントプランニングで約束した ことを実現する責任を持つ。					
<b>ステークホルダー</b> 製品の利用者、出資者、管理 職などの利害関係者。					

- [1] スプリントプランニング第二部へのプロダクトオーナーの参加  
 スプリントプランニング第二部では実現方法及び実現するための計画について話し合うため、常に場にいる必要はない。  
 但し、必要なタイミングで質問に応えられるようにしておくこと。  
 また、開発チームはプロダクトオーナーに対し、どのようにスプリントゴールを達成するかを説明する必要がある。
- [2] デイリースクラムへのプロダクトオーナーの参加  
 スプリントゴールを達成するにあたっての障害事項が頻繁に現れ、プロダクトオーナーの支援が必要な場合などにおいて有効。
- [3] デイリースクラムへのスクラムマスターの参加  
 スクラムマスターは開発チームにデイリースクラムを開催してもらうようにするが、開催する責任は開発チームにある。  
 初期のチームにおいては、タイムボックスの遵守やルールを守らせる役割として入ることもある。  
 開発チームだけでうまくデイリースクラムを行えているのであれば、参加する必要はない。
- [4] スプリントレトロスペクティブへのプロダクトオーナーの参加  
 プロセスの検査と改善が目的のため、POがいると率直な問題点を明らかにできない場合はPO抜きで行う。

- ：参加すべき  
 ×：参加不可  
 △：コンテキストにより参加  
 ⊘：参加しても良いが発言権なし

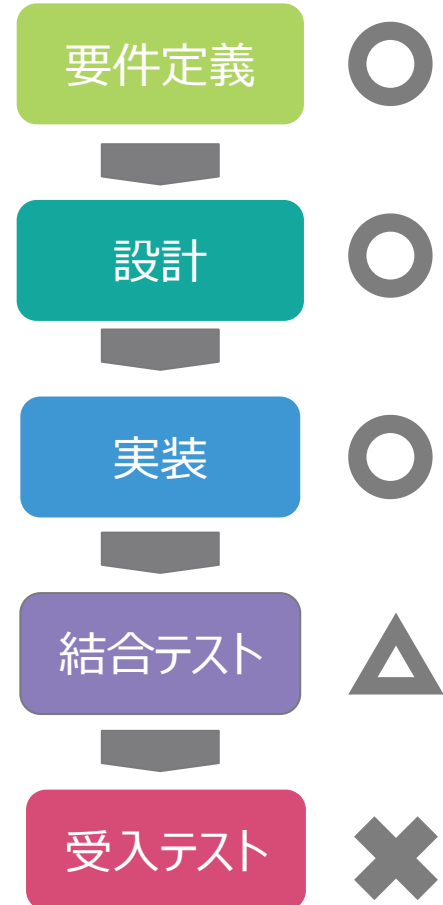
# スクラムと品質

---

## ウォーターフォールでの品質課題

フェーズゲートにより内部品質は担保されるが、作成した製品が本当に欲しいモノだったのかわかるのは受入テストのタイミング。

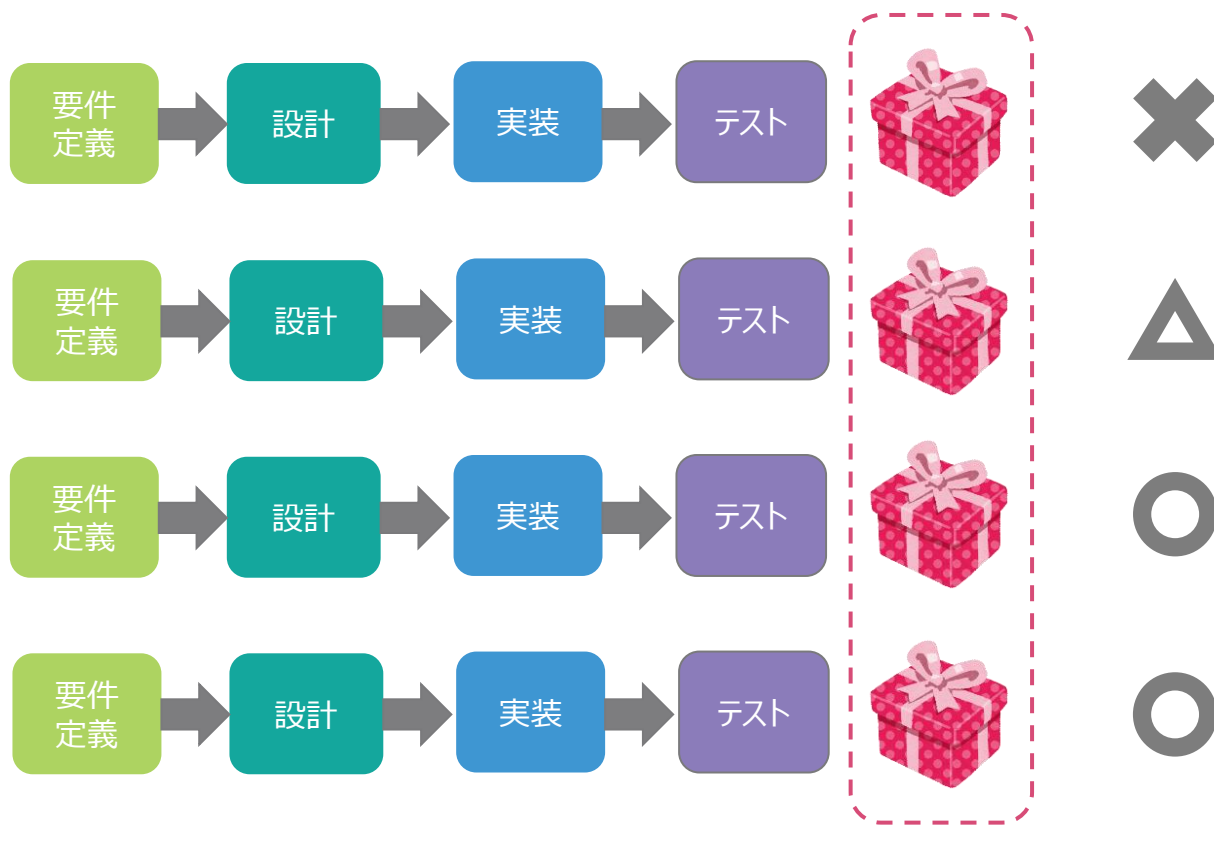
大きなリスクが後半に待っている。



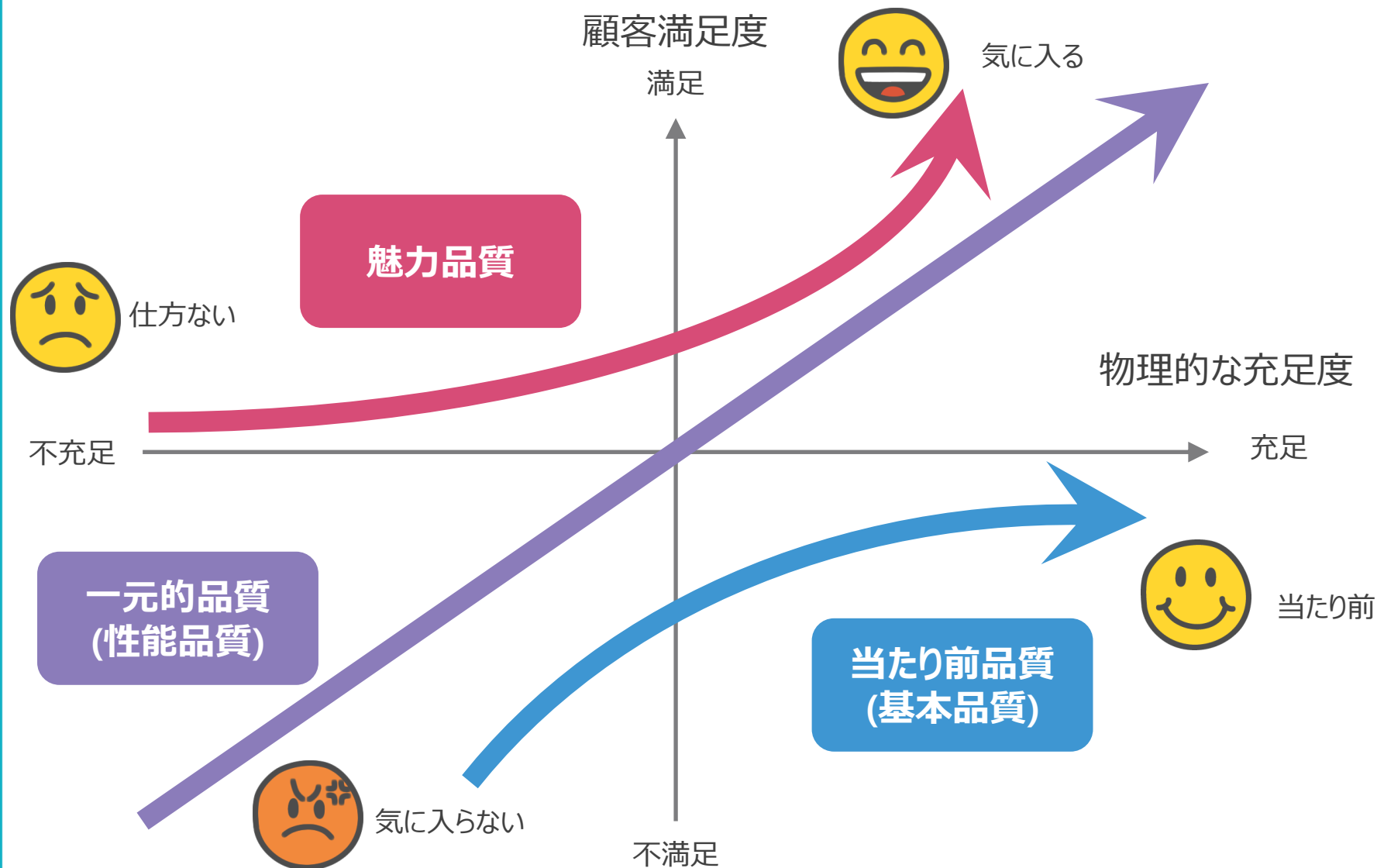
# スクラムでは

スプリントレビューでビジネス要件との適合性を確認

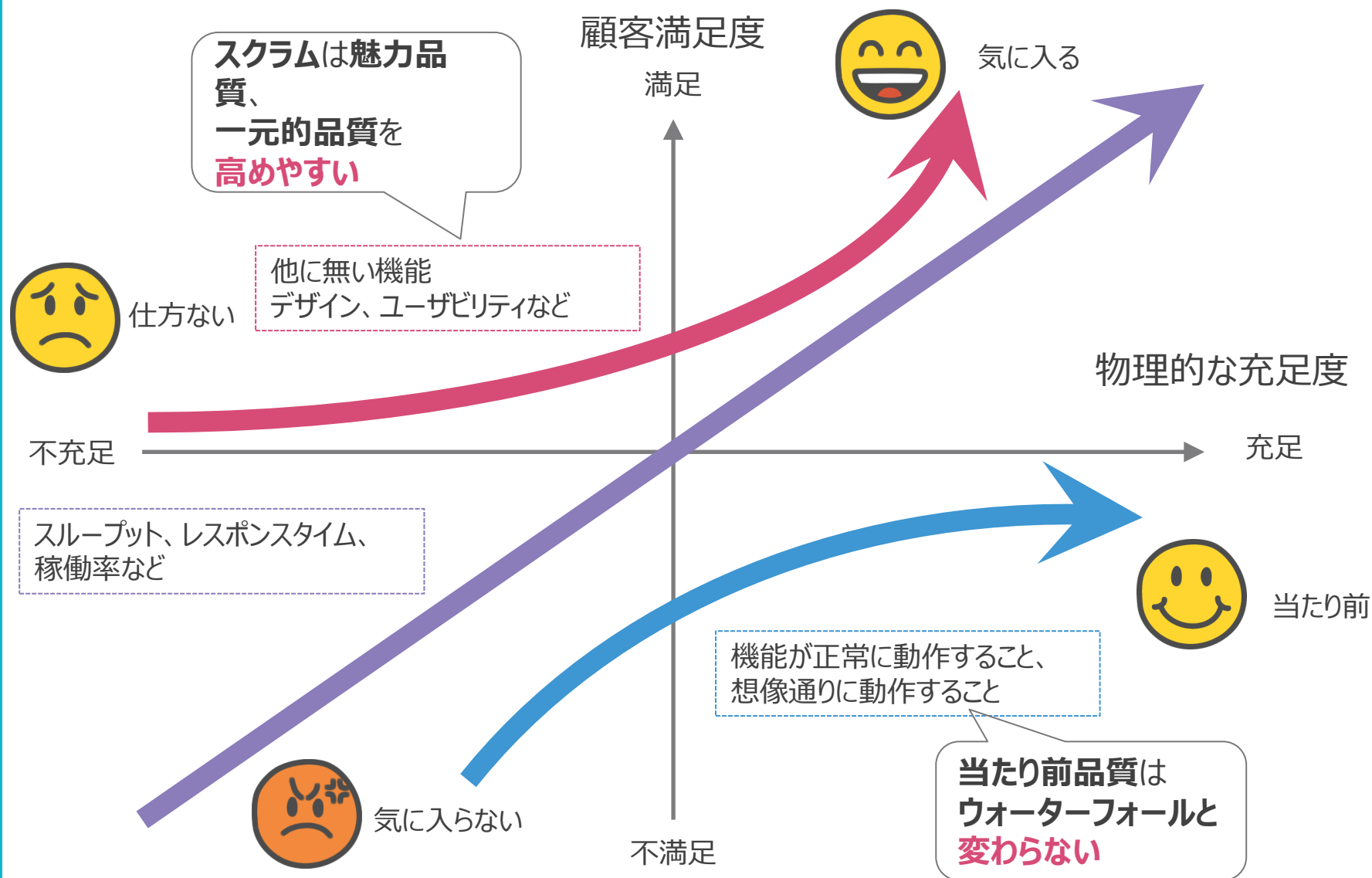
時間



# 狩野モデル



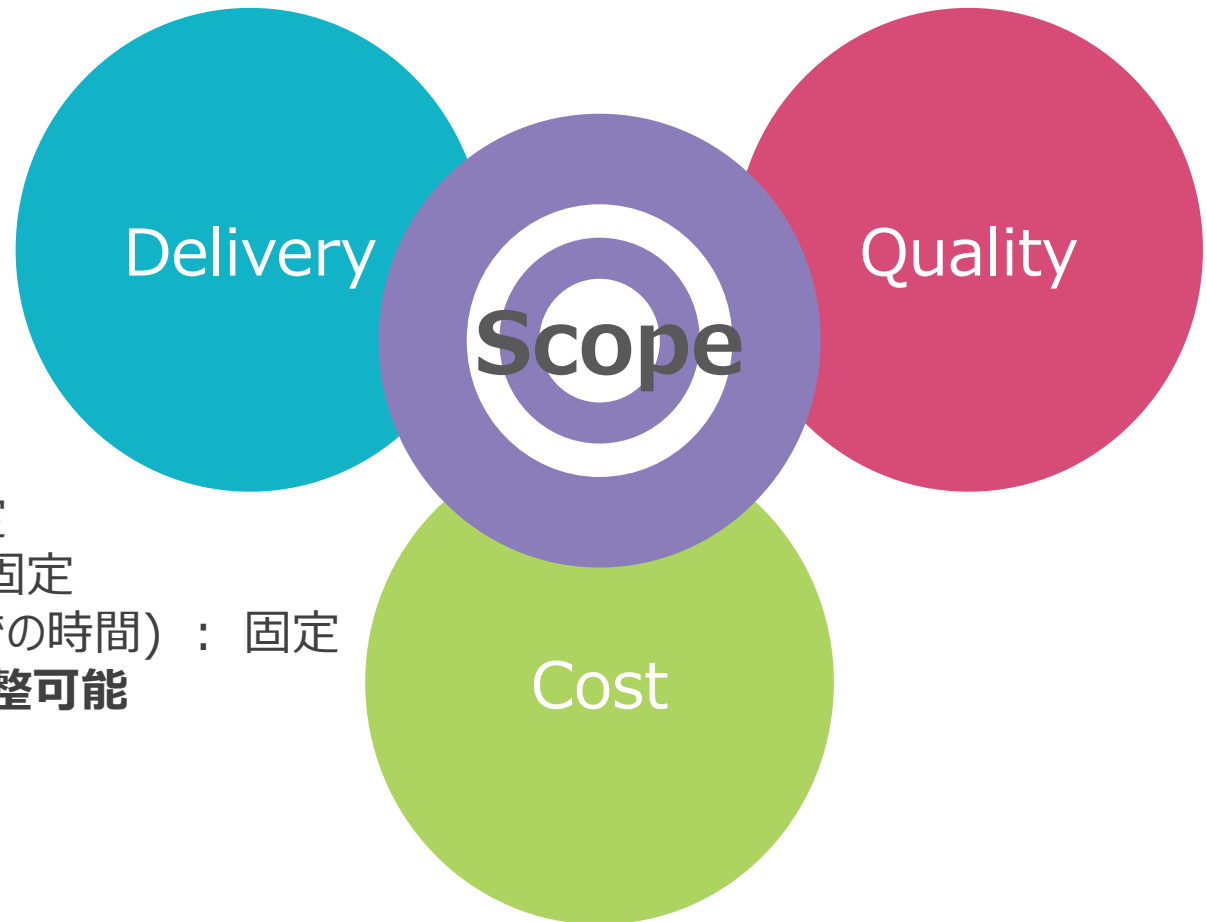
# 狩野モデル





## QCD + S

---

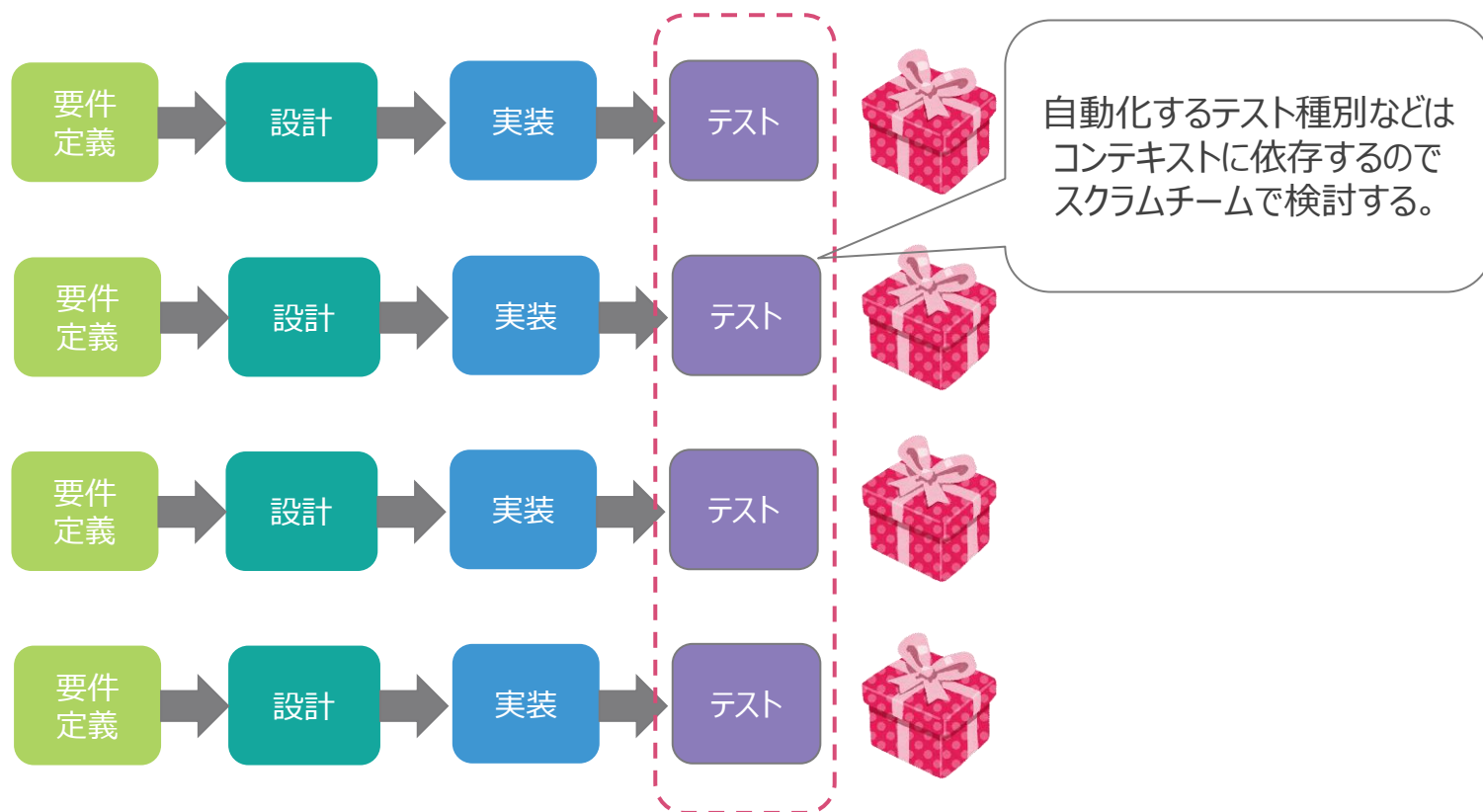


- Quality(品質) : 固定
- Cost(コスト・体制) : 固定
- Delivery(価値提供までの時間) : 固定
- Scope(スコープ) : **調整可能**

# テスト自動化

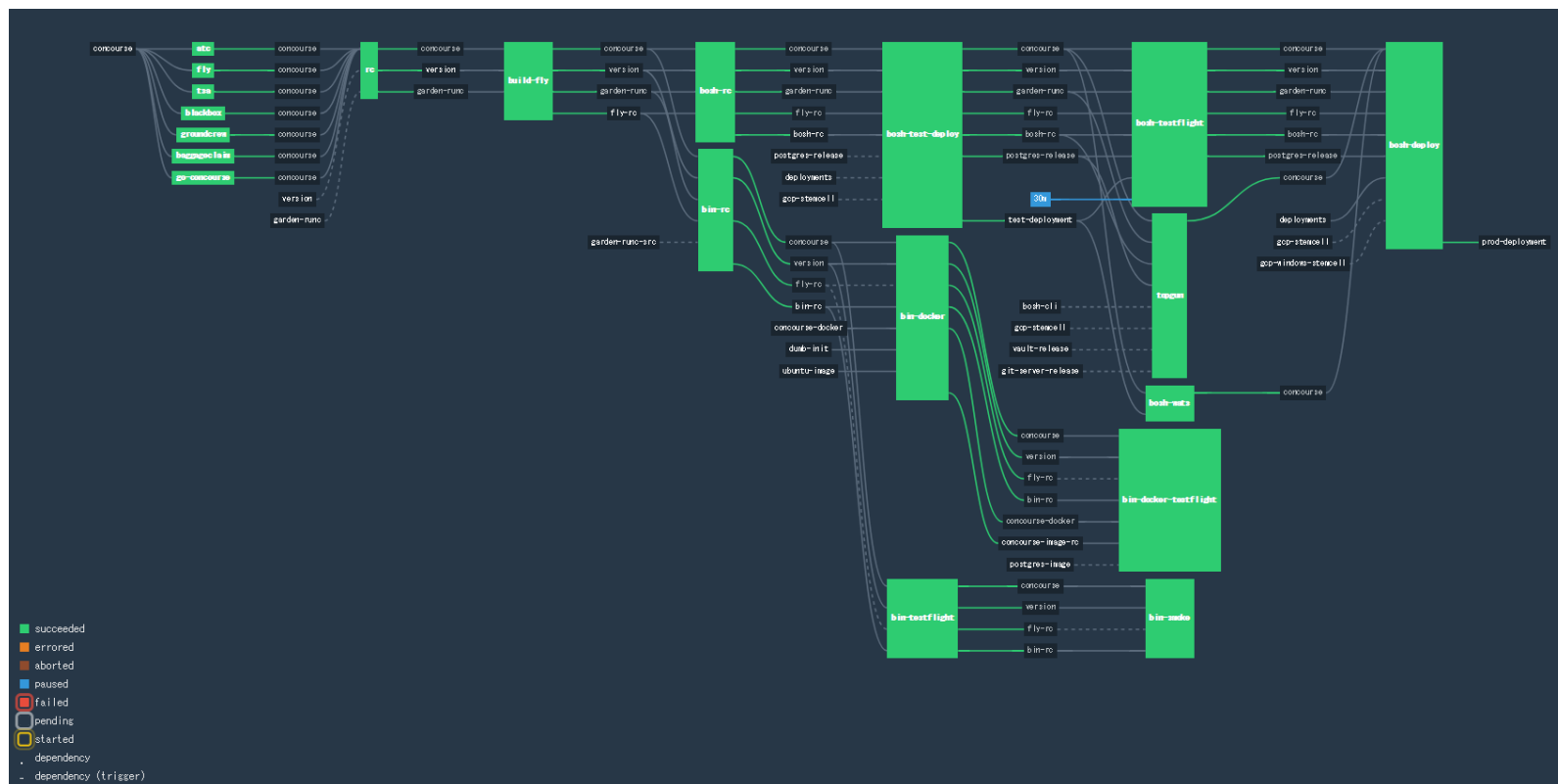
繰り返し開発していくことになるので、テストを行う回数がウォーターフォールより多い。  
前のスプリントで追加した機能のリグレッションテストなどを考えると、**自動化は必須**。

時間



# 継続的インテグレーション/デリバリー

テスト自動化と同様に、頻繁にビルド・デプロイを行うことになるため、自動化を推奨する。



# スクラムとメトリクス

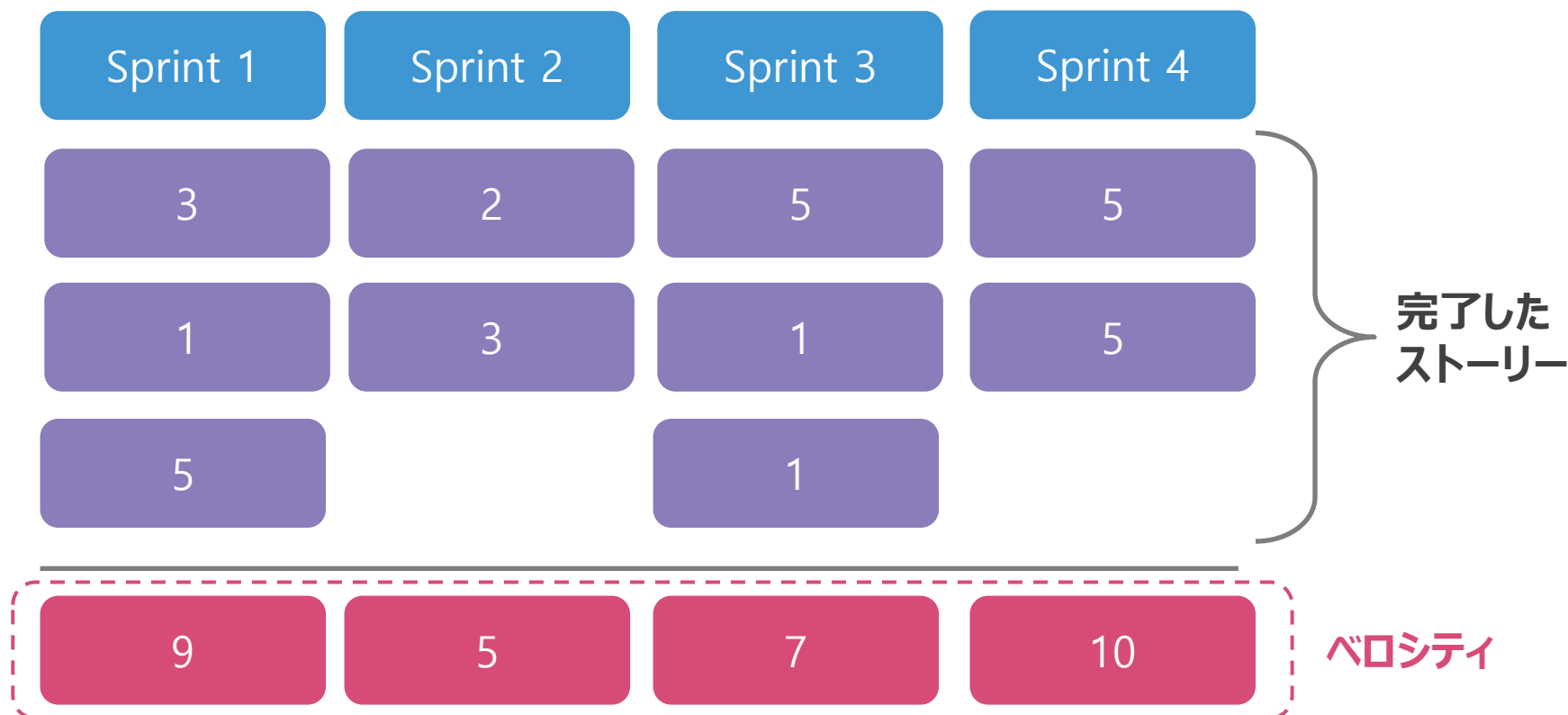
---

# ベロシティ

1スプリントで完了したストーリーポイントの合計。

開発チームの**生産量**を表す。

スプリントプランニングでキャパシティを決める際に参考にする。



# ベロシティの利用

## リリース予定に対して遅延しているかどうか

リリースポイント  $\leq$  (期待 or 実績) ベロシティ  $\times$  残スプリント数

例：  $100 \leq 10\text{ポイント} \times 10\text{スプリント} \rightarrow$  オンスケジュール

$100 \geq 9\text{ポイント} \times 10\text{スプリント} \rightarrow$  遅延

## キャパシティの求め方

キャパシティ = 5スプリント分の有効ベロシティの平均

有効ベロシティは初出のベロシティ、今までの最大、最小を除外したもの。

有効ベロシティが5スプリント分ない場合は、直近のベロシティを採用する。

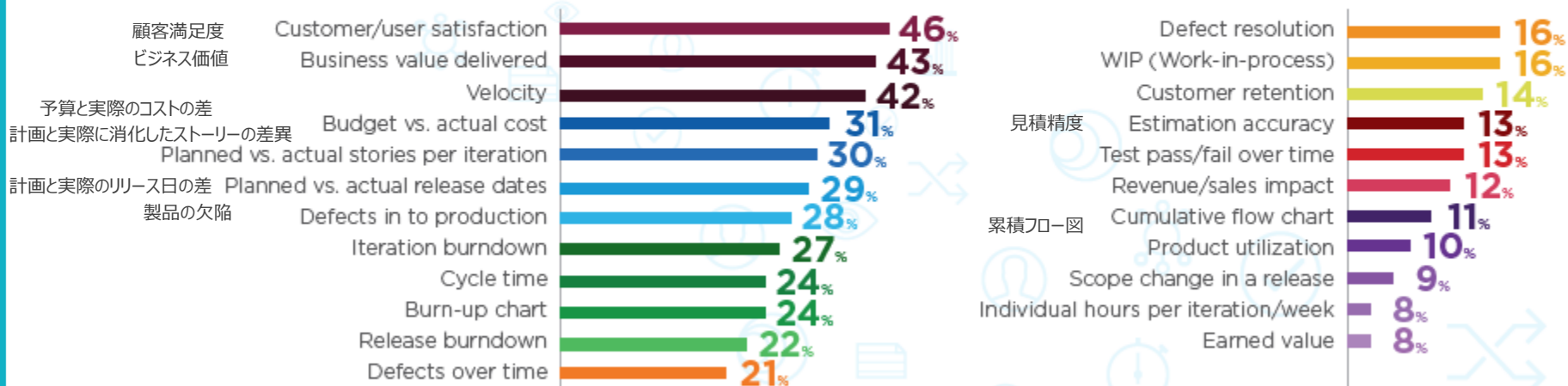


# アジャイルプロジェクトでの成功の測定方法

ビジネス価値が、23%(2016年)から**43%(2017年)**に増加。  
顧客満足度が28%(2016年)から**46%(2017年)**に増加する一方、  
ベロシティが67%(2016年)から**42%(2017年)**に減少。

## How Success Is Measured... with Agile Projects

Business value increased as a cited measure of agile project success from 23% in 2016 to 43% in 2017. Customer/user satisfaction increased from 28% in 2016 to 46% in 2017 while velocity had been the number one measure of an agile project's success decreased from 67% in 2016 to 42% in 2017. Iteration burndown also went down from 2016 (51%) to 2017 (27%).



\*Respondents were able to make multiple selections.

# スクラム・アジャイルに対するよくある誤解

---



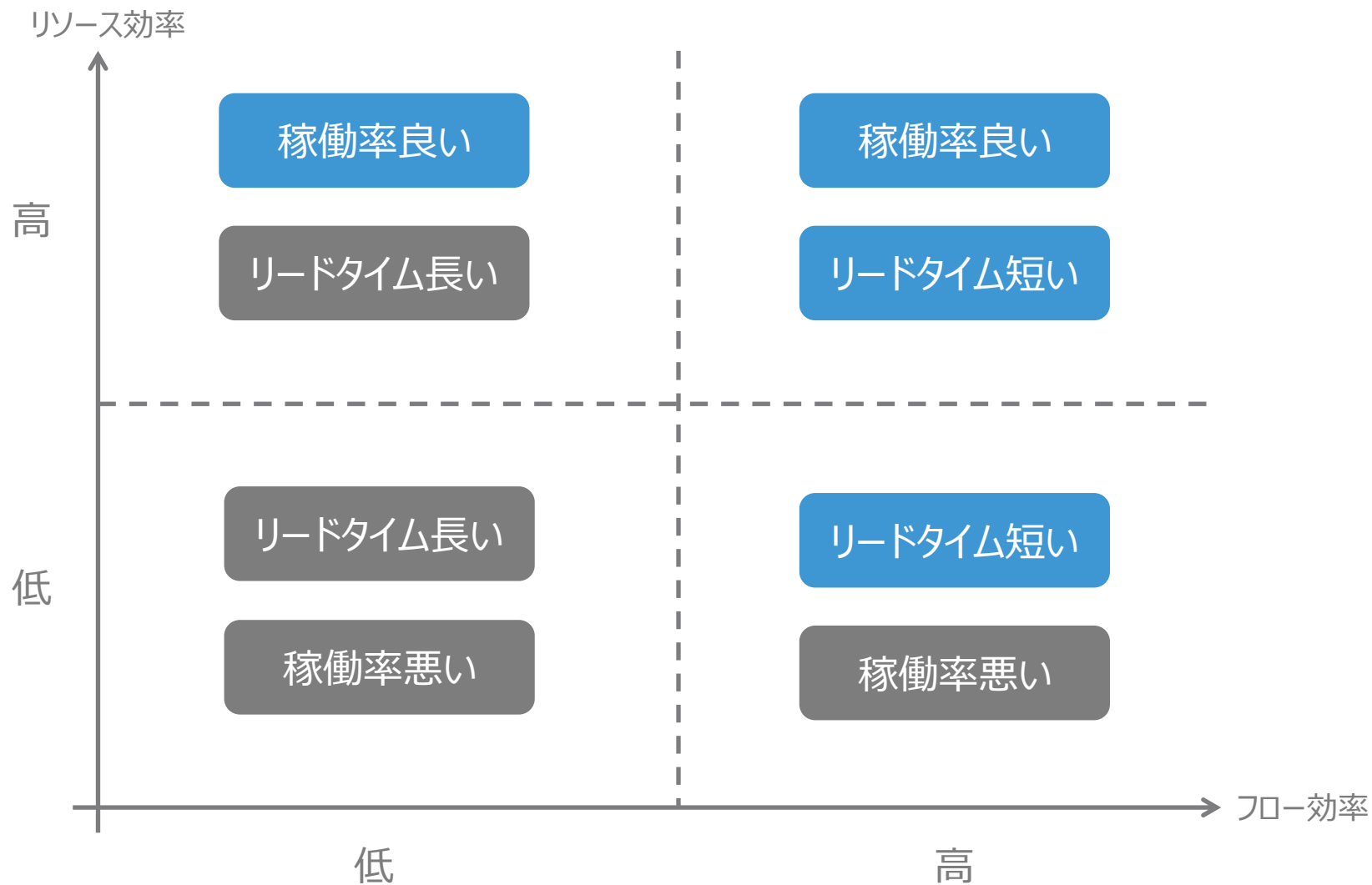
# スクラムは生産性が高い

決まりきったモノを作るのであれば、ウォーターフォールの方が生産性は高い。複雑で変化が激しく、市場やユーザの反応を見なければ正解がわからない場合などは、スクラムが向いている。市場投入までのリードタイムはスクラムの方が早い。

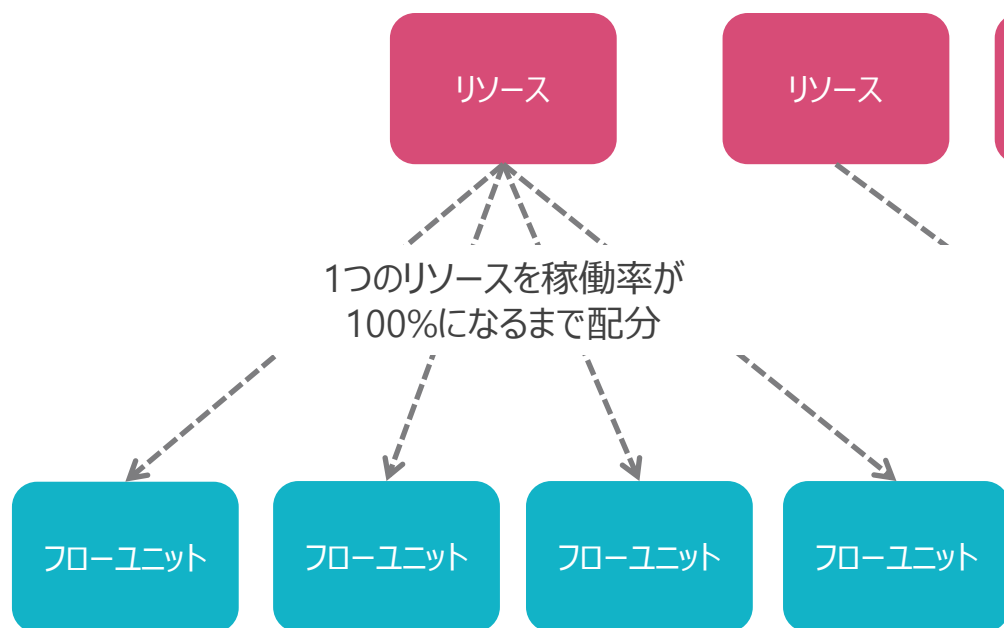
市場投入までのリードタイム =  $P + (S + Q + W)$

要素	定義	システム開発
セットアップタイム(準備)	準備時間	計画、設計に掛かる時間
プロセスタイム	加工時間、移動時間	実装に掛かる時間、デリバリーに掛かる時間
キュータイム	リソースの制約から待っている時間	全員が作業中で、タスクが着手されていない時間
ウェイトタイム	依存するタスクの待ち時間	依存モジュールの完成待ち テスト対象のデリバリー待ちなどをしている時間

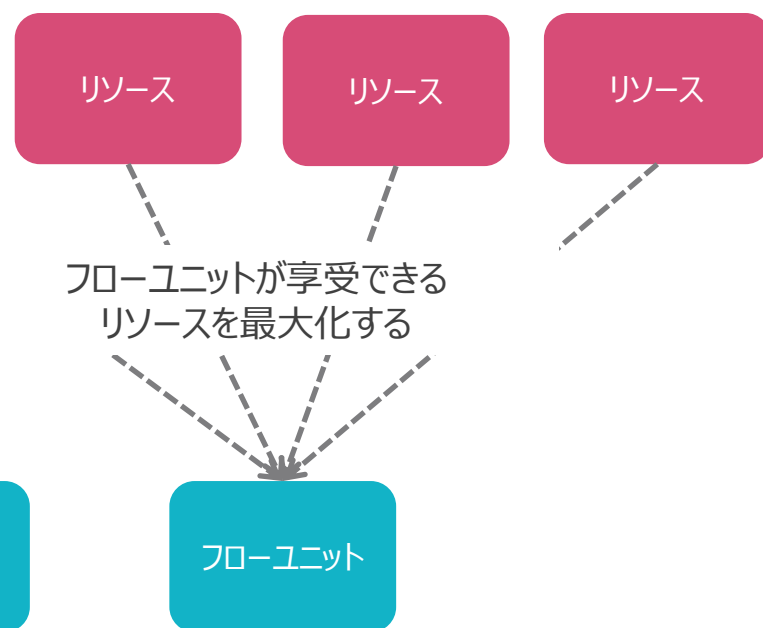
# リソース効率とフロー効率



# リソース効率とフロー効率

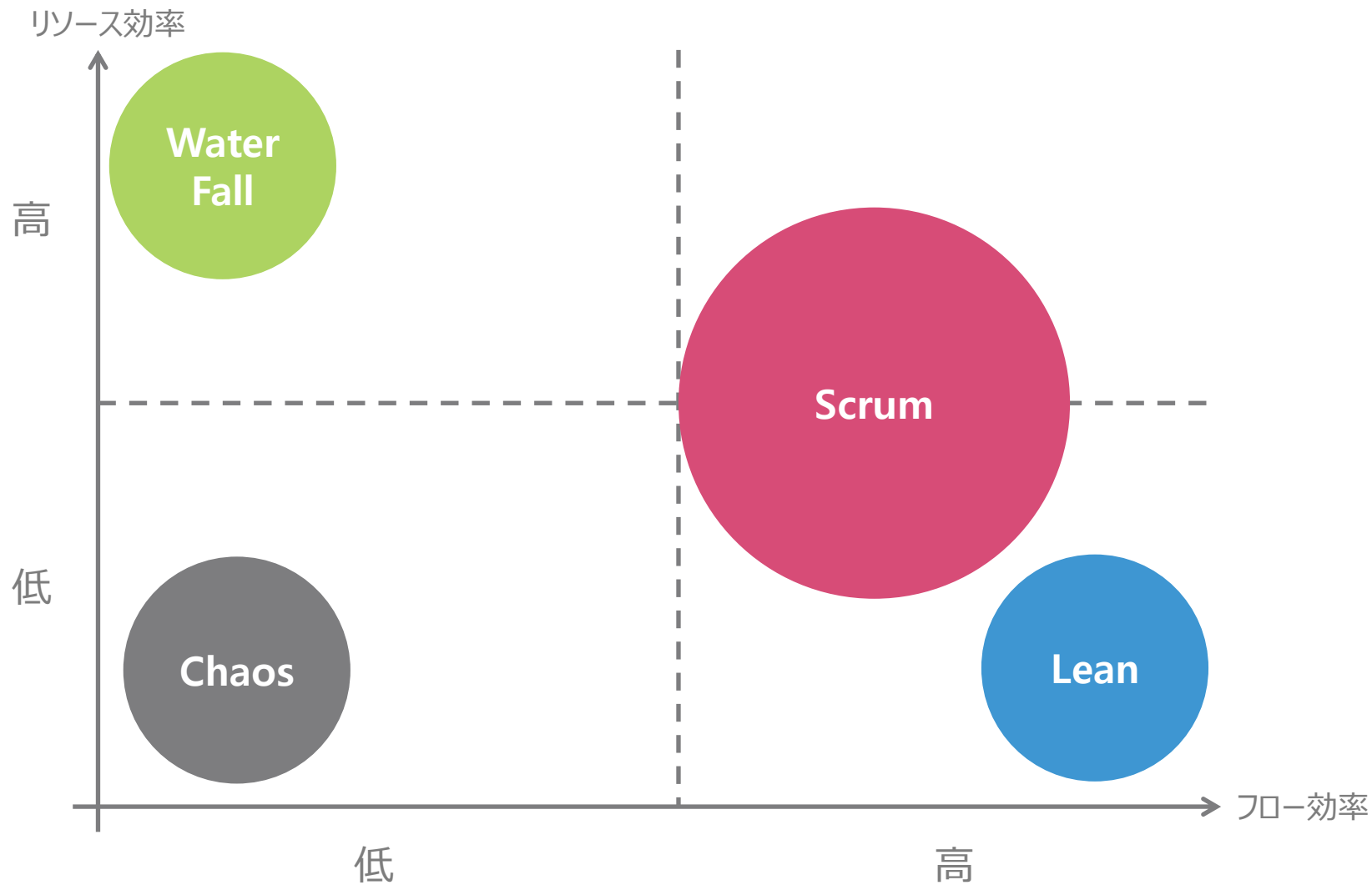


例： マルチタスク

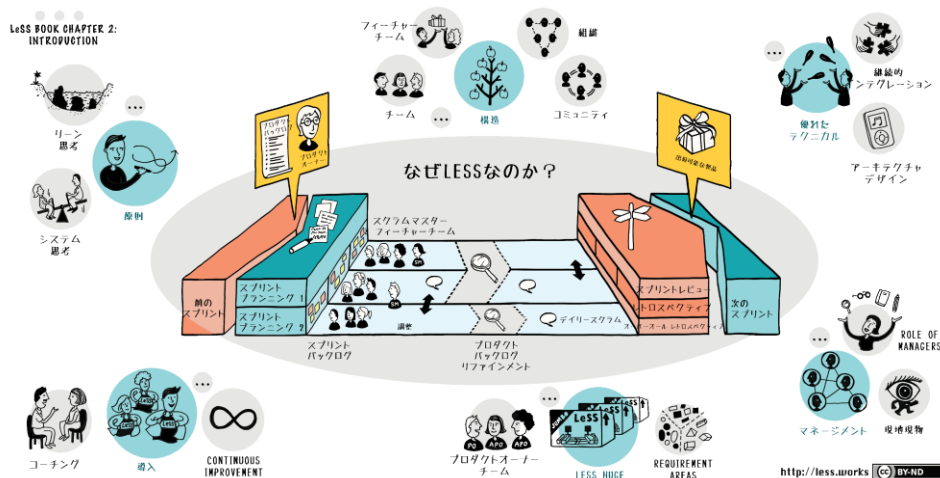


例： ペアプログラミング

# リソース効率とフロー効率



# スクラムは大規模に向かない

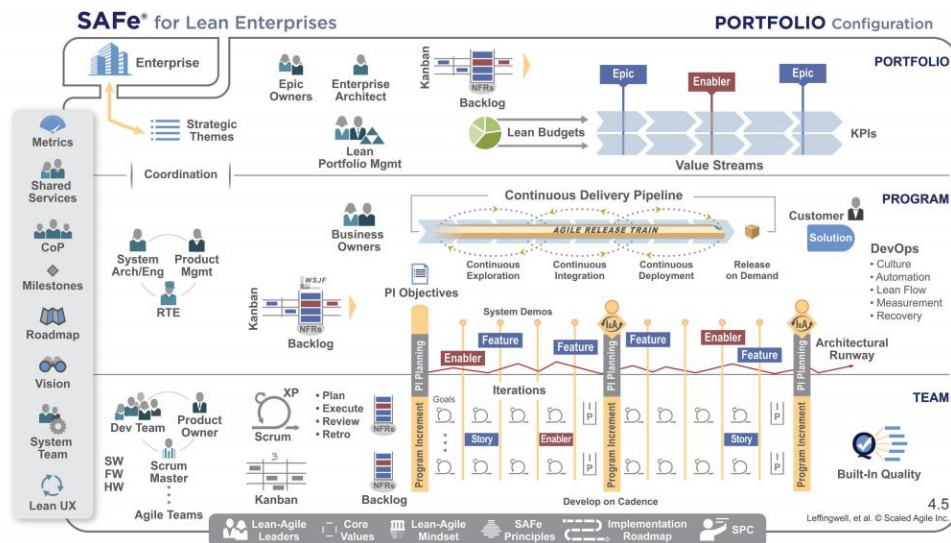


<https://less.works/jp/>

企業規模でアジャイルを実現するためのもの。  
8-12週のサイクルで反復する。  
1リリースに対して5-15チーム(50-125人)が存在する。

2-8チームでの開発向け  
1人のスクラムマスターは1-3チーム担当できる  
1つのプロダクトに対して1つのプロダクトバックログ、  
1人のプロダクトオーナー  
各チームのスプリント周期は同一

8チーム以上はLeSS Hugeというフレームワークがある。

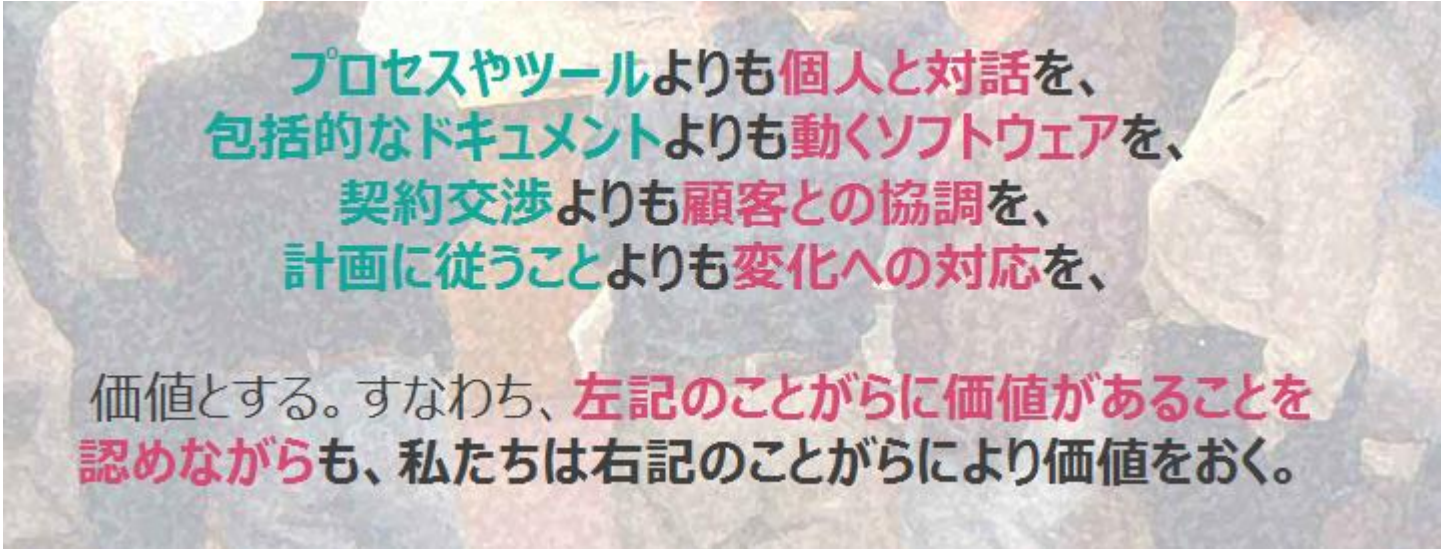


<http://www.scaledagileframework.com/>

SAFe® PROVIDED BY SCALED AGILE

# スクラムはドキュメントを作らない

包括的なドキュメントよりも動くソフトウェアに価値を置いている。  
必要なドキュメントは、当然作成する。

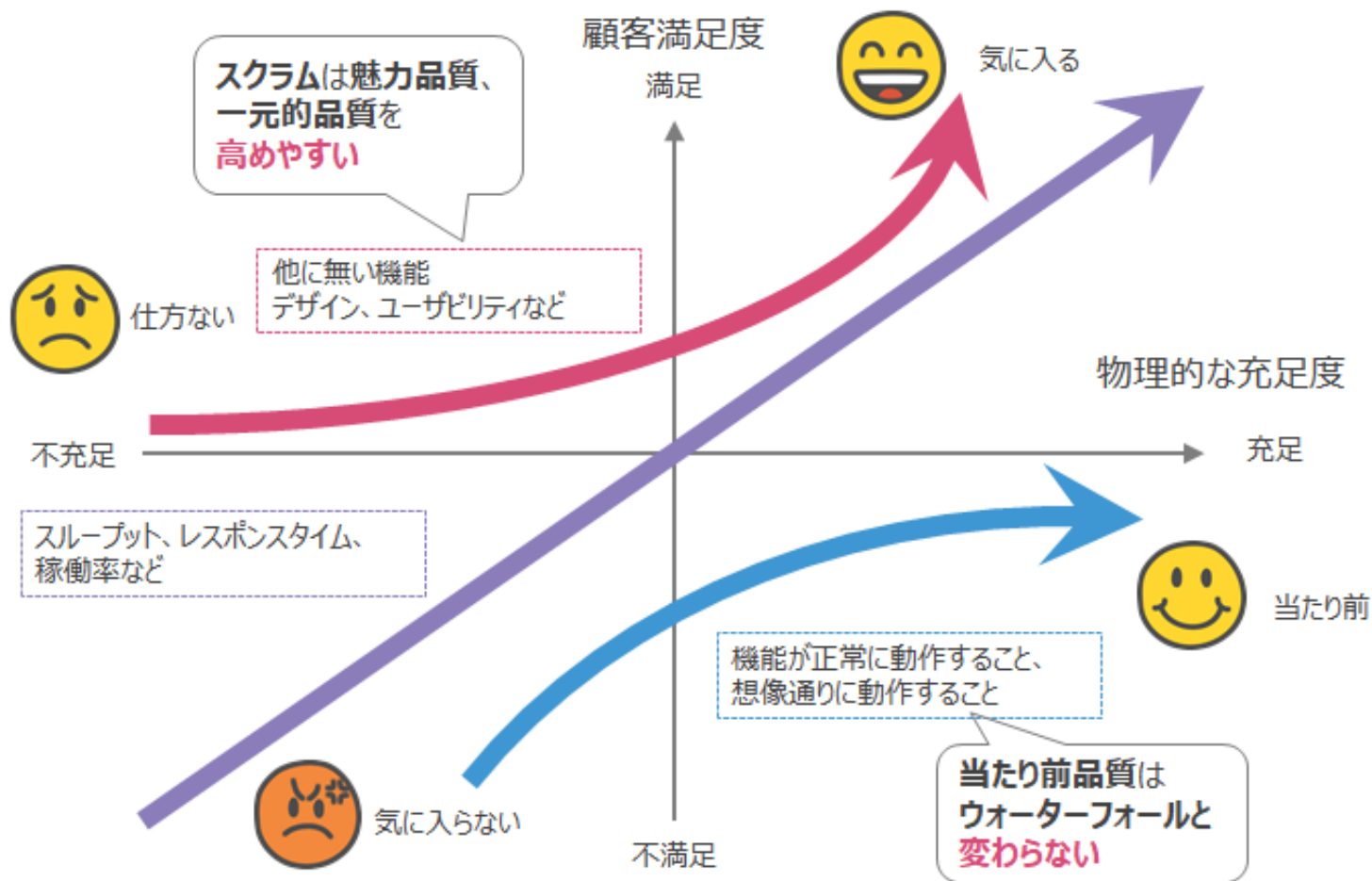


プロセスやツールよりも**個人と対話**を、  
包括的なドキュメントよりも**動くソフトウェア**を、  
契約交渉よりも**顧客との協調**を、  
計画に従うことよりも**変化への対応**を、

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。

# スクラムは品質がウォーターフォールに劣る

開発プロセスによってテスト種別、テストケースは変わらない。  
よって、基本品質に差が出ることは無い。



# Appendix

---

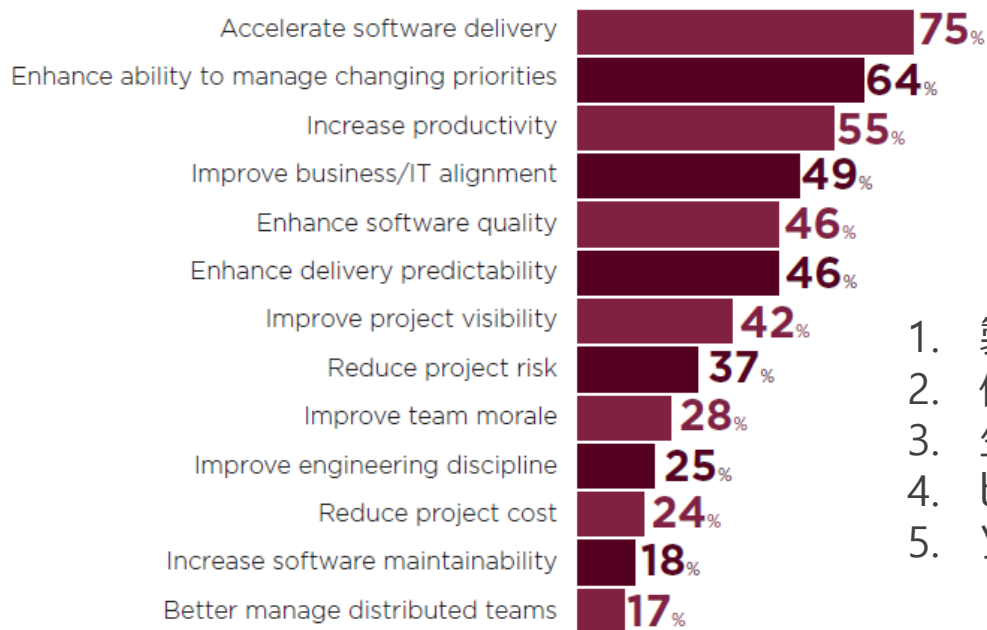


# アジャイルの採用理由

## Reasons for Adopting Agile

製品のデリバリーの加速が更に重視されるように。

The reasons stated for adopting agile follow a similar ranking as in the previous year though we did see the biggest change in responses in accelerate software delivery (75% compared to 69% last year), enhancing delivery predictability (46% compared to 30% last year), improving IT/Business alignment (49% compared to 42% last year), and reducing project cost (24% compared to 18% last year).



1. 製品のデリバリーの加速
2. 優先順位の変更管理能力を高める
3. 生産性を高める
4. ビジネスとITの連携の改善
5. ソフトウェアの品質を高める

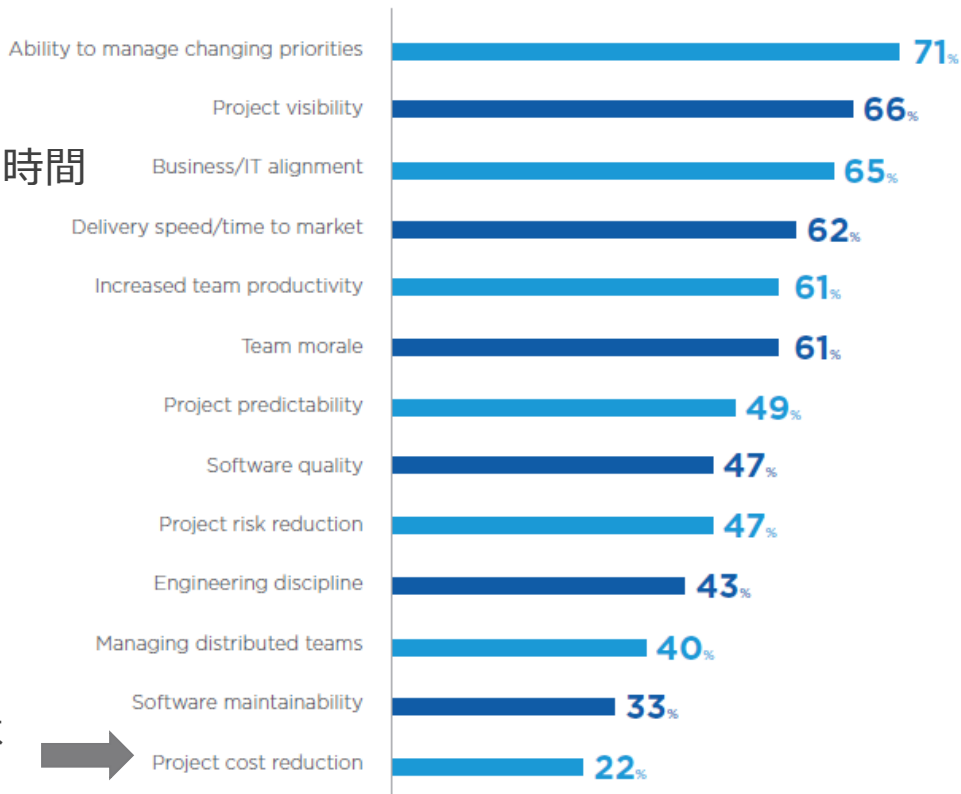
\*Respondents were able to make multiple selections.

# アジャイルを導入するメリット

1. 優先順位の変更管理
2. プロジェクトの可視性
3. ビジネスとITの連携
4. デリバリー速度/市場投入までの時間
5. チームの生産性向上

プロジェクトのコスト削減と答えている数は  
意外と少ない

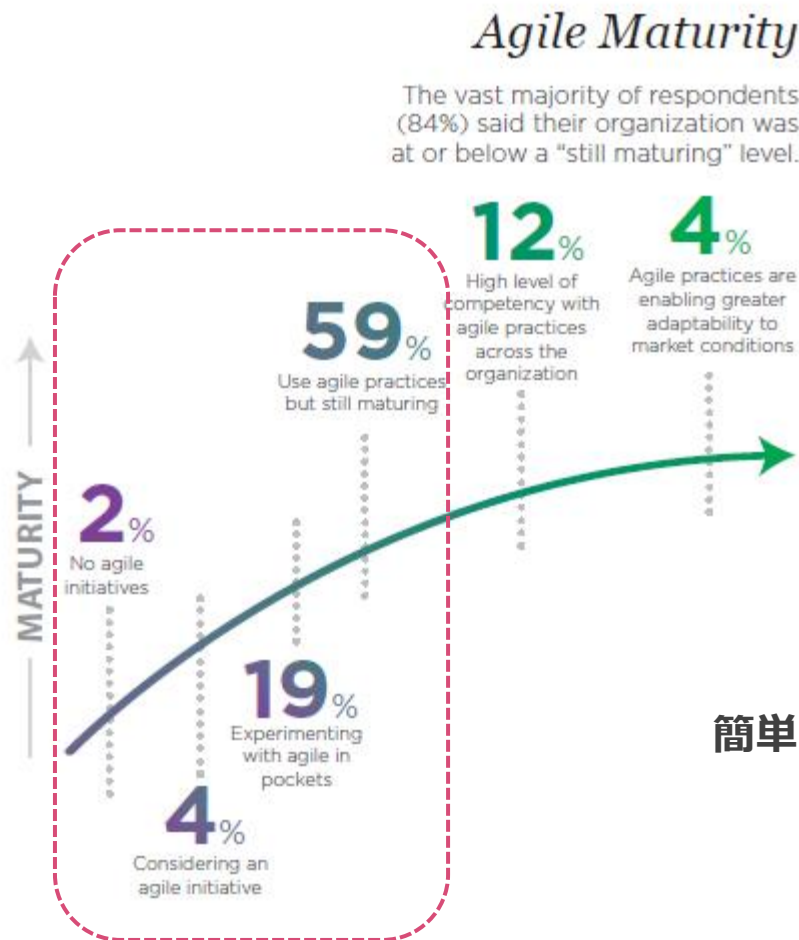
*Benefits of Adopting Agile*  
By implementing agile, respondents cited seeing improvements in the following areas:



\*Respondents were able to make multiple selections.

# アジャイルの成熟度

80%以上がまだ成熟しきっていないと回答



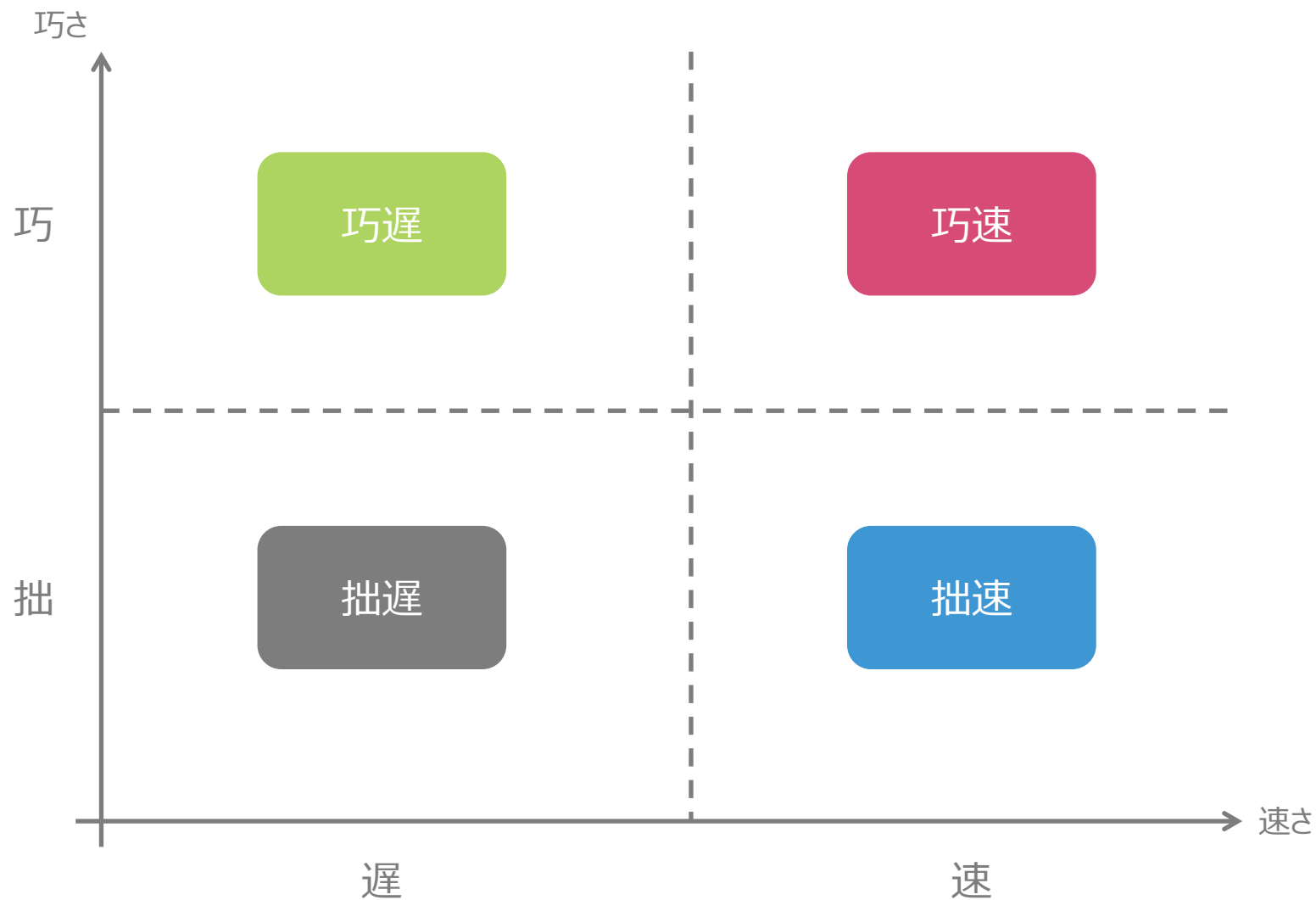
簡単には習得できない

# 構成管理及びリリース管理の成熟度モデル

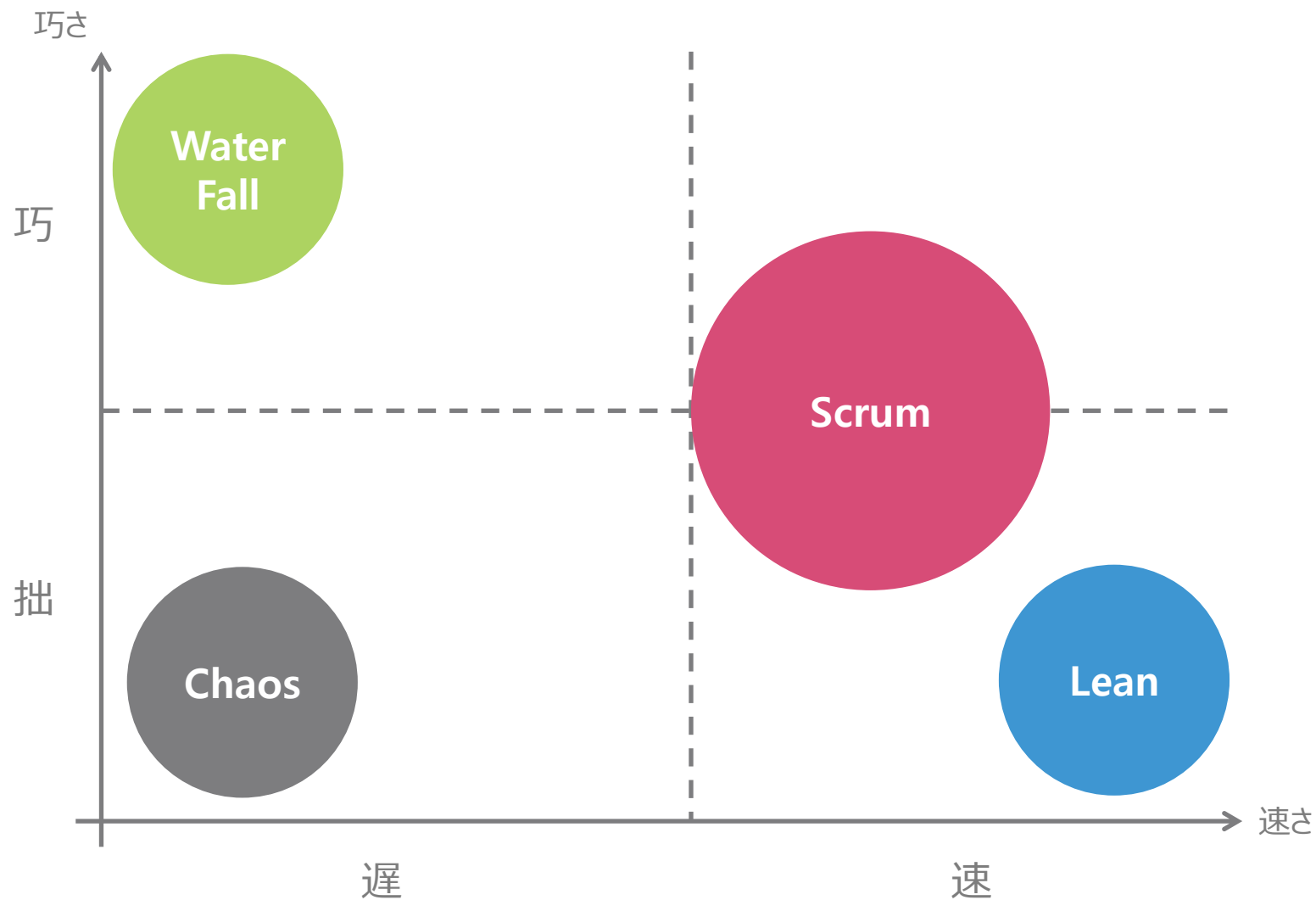
プラクティス	ビルド管理および 継続的インテグレーション	環境および デプロイメント	リリース管理および コンプライアンス	テスト	データ管理	構成管理
レベル3 - 最適化： プロセスの改善に注力する	チームで定期的に話し合いの場を持ち、統合時の問題やその自動化による解決、素早いフィードバック、そしてよりよい可視化について議論する。	全ての環境がうまく管理されている。プロビジョニングは完全に自動化。仮想化を適切に活用する。	運用チームとデリバリーチームが協力し、リスク管理やサイクルタイム削減を行う。	本番環境への変更の取り消しはめったに発生しない。問題があればすぐに見つかり、すぐに修正される。	リリースのたびに、データベースのパフォーマンスやデプロイメントプロセス自体についてのフィードバックを得る。	変更管理ポリシーを常に検証し、効率的な共同作業や素早いデプロイができていないかを確かめる。また、変更管理プロセスの可監査性もチェックする。
レベル2 - 定量的な管理： プロセスが計測可能で制御されている	ビルドメトリクスを収集して可視化し、それに基づいて作業する。ビルドを壊れたままにしない。	統合したデプロイ管理、リリースやリリース取り消しの手順もテストしている。	環境はアプリケーションの健康状態を監視し、能動的に管理している。サイクルタイムを監視している。	品質のメトリクスとその傾向を追跡する。非機能要件を定義し、計測する。	データベースの更新やロールバックはデプロイのたびにテストされる。データベースのパフォーマンスを監視、最適化する。	開発者は、少なくとも一日一度はメインラインにチェックインする。ブランチはリリース作業の時にだけ使う。
レベル1 - 一貫している： 自動化されたプロセスが、アプリケーションのライフサイクル全体に適用される	自動ビルドと自動テストのサイクルを、変更がコミットされるたびに実行する。依存関係を管理する。スクリプトやツールを再利用する。	ソフトウェアのデプロイは完全に自動化され、ボタンを押すだけで完結する。すべての環境に対して同じ手順でデプロイする。	変更管理とその承認プロセスが定義され、それを守っている。規約を順守している。	ユニットテストや受け入れテストを自動化する。受け入れテストはデスターが書く。テストが開発プロセスに組み込まれる。	データベースへの変更は、デプロイメントプロセスの一環として自動的に行う。	ライブラリや依存関係を管理する。バージョン管理システムの利用ポリシーは、変更管理プロセスで定義する。
レベル0 - 繰り返し可能： プロセスは文書化され、一部は自動化されている	普段のビルドやテストを自動化する。すべてのビルドは、ソース管理システムを使って自動化された手順で再現できる。	一部の環境ではデプロイを自動化する。新しい環境を手軽に作成する。すべての構成管理情報を外に出してバージョン管理する。	面倒で頻度も低く、信頼できないリリース。リリース要件に関するトレーサビリティも限定的。	ストーリーの開発の一環として自動テストを書く。	データベースへの変更は自動化したスクリプトで行い、スクリプトはアプリケーションとともにバージョン管理する。	バージョン管理システムを使って、ソフトウェアの作成に必要なものをすべて管理する。ソースコードや設定ファイル、ビルドやデプロイ用スクリプト、データのマイグレーションなど。
レベル-1 - リグレッションエラー多発： プロセスは繰り返せず管理も貧弱、そして対処療法を行っている	ソフトウェアのビルド手順が手動である。成果物やビルド結果の管理をしていない。	ソフトウェアのデプロイ手順が手動である。バイナリが環境に依存する。環境の配布が手動である。	リリース頻度が低く、しかも信頼できない。	開発をした後に手作業でのテストを実施する。	データのマイグレーションは、バージョン管理されておらず、手動で操作する。	バージョン管理システムを使っていない。あるいは使っていても滅多にチェックインしない。

David, Farley ; Jez, Humble. 継続的デリバリー: 信頼できるソフトウェアリリースのためのビルド・テスト・デプロイメントの自動化. 和智 右桂訳 ; 高木 正弘訳. KADOKAWA/アスキー・メディアワークス. 2012.

# 拙速と巧遅



# 拙速と巧遅



# アジャイルをスケールさせるには

## *Top 5 Tips for Success with Scaling Agile*

Respondent indicated the most valuable in helping them scale agile practices were:



\*Respondents were able to make multiple selections.

1. 内部のアジャイルコーチ
2. チーム間での同じプラクティスとプロセス
3. チーム間での共通のツールの実装
4. 外部のアジャイルコンサルタント/トレーナー
5. 経営陣の後援