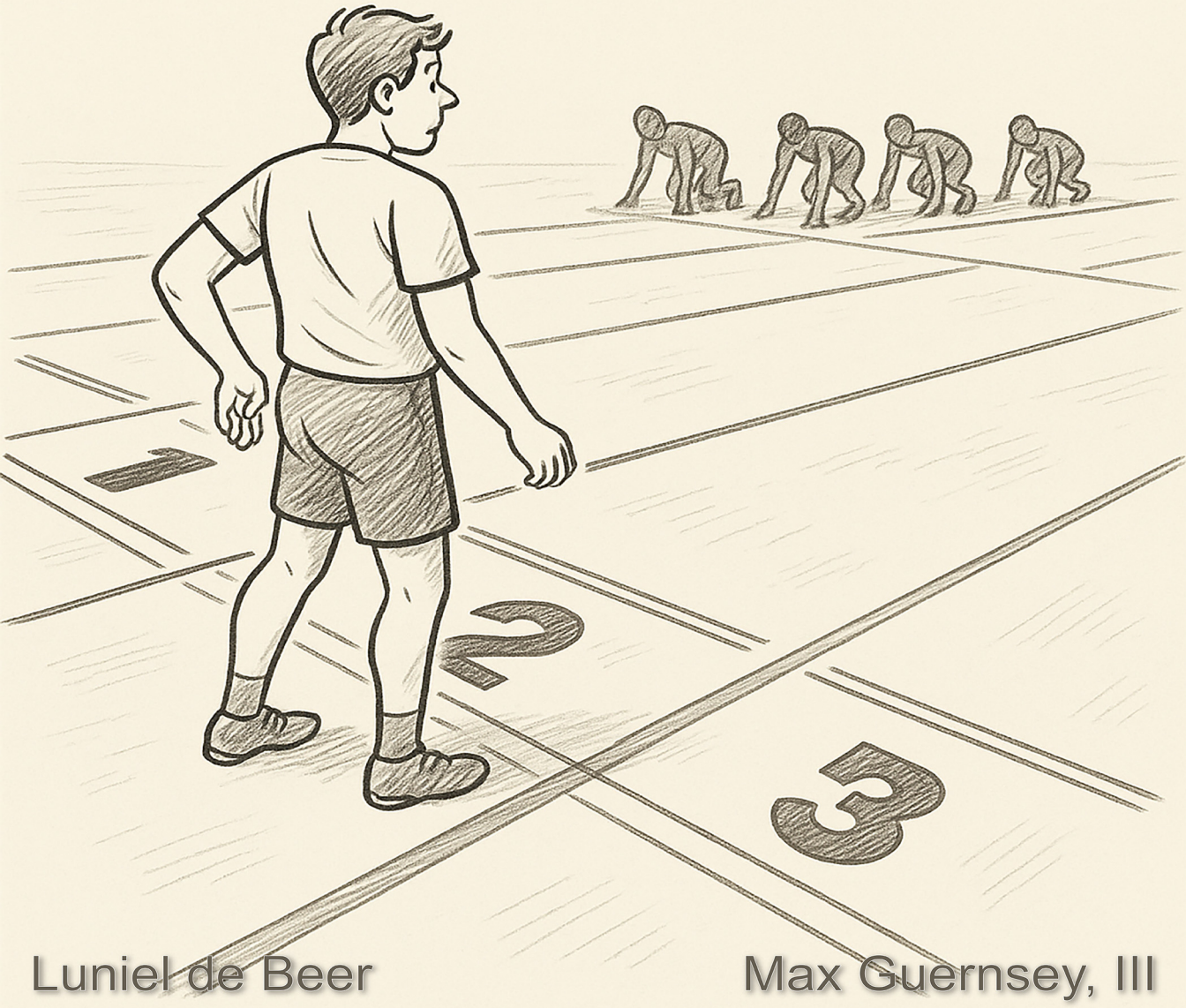
# READY

## WHY MOST SOFTWARE PROJECTS FAIL AND HOW TO FIX IT




Luniel de Beer

Max Guernsey, III

日本語版

# この本をツイートしよう！

Twitterでこの本の情報を共有して、Luniel de Beer と Max Guernsey, III をサポートしてください！

この本のおすすめツイート文：

ソフトウェアデリバリーにおけるやり直し作業、持ち越し作業、不一致を解消したいソフトウェアリーダーとチームのための書籍「Ready」を購入しました。#CodeReady

この本のおすすめハッシュタグは#CodeReadyです。

以下のリンクをクリックして、このハッシュタグで他の方々の感想をチェックしてください：

#CodeReady

# Ready (日本語版)

なぜソフトウェアプロジェクトの多くは失敗するのか？
その改善策を探る

Luniel de Beer と Max Guernsey, III

本書はhttps://leanpub.com/ready-jaでお求めいただけます

このバージョンは 2025-10-22 に公開されました


これは Leanpub の書籍です。Leanpub は著者や出版社に Lean Publishing プロセスを提供しています。Lean Publishing とは、軽量なツールを使用して進行中の電子書籍を公開し、読者からのフィードバックを得ながら、適切な書籍になるまで改善を重ね、その後の展開を図っていく手法です。



ヨハン・ファン・アールトの追悼の意を込めて。私の情熱を見出し、本物のプログラミングの世界へと導き、新しい故郷への道を示してくれた方へ。そして両親へ。最初のクライアントから米国での最初の家まで、すべてを可能にしてくれた揺るぎない支援に感謝を込めて。

—— ルニエル

私の家族へ。共に日々を歩む愛する人たちへ。

—— マックス

# Contents

本書について . . . . . i

対象読者 . . . . . ii

本書の使い方 . . . . . iii

著者について . . . . . iv

はじめに . . . . . v

## パート I: 何かが足りない . . . . . 1

章 1: 隠された問題 . . . . . 2

章 2: 基礎が欠如することのコスト . . . . . 14

章 3: 要件成熟化フロー（RMF）の紹介 . . . . . 26

章 4: アジャイルなのか？ . . . . . 28

## パート II: 準備態勢のための場づくり . . . . . 29

章 5: 最初の拡張 . . . . . 30

章 6: なぜ人々はこれを行わないのか？ . . . . . 31

章 7: 明示的な準備作業（RMF 1） . . . . . 33

章 8: RMF 1 の効果 . . . . . 35

章 9: RMF 1 を実践に移す . . . . . 36

# パート III: 作業完了の承認プロセス . . . . . 38

章 10: 次なる要件 . . . . . 39

章 11: 一般的な人々の行動 . . . . . 41

章 12: 完了の定義を定義する . . . . . 43

章 13: カスタマイズされた完了の定義（RMF 2） . . . . . 45

章 14: RMF 1・2 との生活 . . . . . 49

章 15: RMF 2 のインストール . . . . . 51

# パート IV: ゲーティング実装 . . . . . 53

章 16: 最終要件 . . . . . 54

章 17: 準備完了の定義の背景 . . . . . 56

章 18: 準備完了の定義を定義する . . . . . 58

章 19: カスタマイズされた準備完了の定義（RMF 3） . . . . . 60

# パート V: 統合 . . . . . 64

章 20: ほとんどの締切は重要ではない . . . . . 65

章 21: 能力 1：要件成熟化フロー . . . . . 68

章 22: スクラムにおける RMF を用いた作業と情報の流れ . . . . . 70

章 23: RMF の影響 . . . . . 72

章 24: RMF への移行 . . . . . 73

章 25: すべてはあなた次第 . . . . . 75

# パート VI: リソース . . . . . 76

付録 A：スクラムが問題なのではない . . . . . 77

付録 B: Synapse フレームワーク ™ . . . . . 79

付録 C：RMF 1 に対する一般的な異議と障害 . . . . . 81

付録 D：DoD 初期基準リスト . . . . . 82

付録 E：DoR スターター基準リスト . . . . . 83

索引 . . . . . 84

# 本書について

*Ready* は、**不完全な納品**、**慢性的な手戻り**、そして**不明確な要件**に悩むソフトウェア開発に関わるすべての人のための本です。

チームの実行スキル向上への投資、プロセスフレームワークの改善実施、またはコードの改修を試みても、さらなる改善が必要と感じているかもしれません。

これは、ほとんどのソフトウェア開発チームにとって主な制約が、チームスキルではなく、要件の成熟度にあるためです。**適切なスキルを持つ成熟したチームでさえ**、未熟な要件に基づいて作業をする際には苦戦します。

*Ready* では、RMF（要件成熟化フロー）を紹介します。これは、既存のプロセスを置き換えることなく、プロダクトとエンジニアリングの連携を図るための実践的で体系的なアプローチです。

Scrum、かんばん、あるいは独自のプロセスのいずれを使用していても、RMF は**スコープの安定化**、**持ち越し作業の排除**、そして**本当に重要なものの提供**を支援します。

チームが「ほぼ完了」の境界線で行き詰まりを感じているなら、本書は**その悪循環を断ち切り**、チームを**確実に**前進させる方法を示します。

# 対象読者

本書は文字通り、ソフトウェア開発に関わるすべての人のための本です。エンジニアからプロダクトマネージャーまで、個人の貢献者から経営幹部まで、すべての方々が対象です。

ソフトウェア開発に関わっており、あなたが所属するチームまたは協働するチームに以下のような問題が一つでもある場合、本書はあなたのためのものです：

- 作業が頻繁にイテレーションから次のイテレーションへ持ち越される
- 実装チームが動く標的を追いかけているように感じる
- 作業が長期間オープンのままになっている
- 作業が完了としてマークされているが、実際には本当に完了していない
- 完了した作業が期待と一致していない
- 作業が定期的に多くの不具合を生み出す
- 大量の作業を定期的にやり直す必要がある

これらの問題のいずれかに心当たりがある場合、*Ready* がお役に立てるでしょう。

# 本書の使い方

本書は実践的であることを目的として設計されています。これは理論的な論文や戦略的なプレゼンテーション資料ではなく、実際のクライアントワークに基づき、実際の納品プレッシャーの下でフィールドテストされた RMF（要件成熟化フロー）を導入するための実践マニュアルです。

各章は順序立てて書かれていますが、RMF 自体はモジュール式です。以下の 3 つの基本的な実践から構成されています：

- RMF 1: 共通理解を得るための協働
- RMF 2: カスタマイズされた完了の定義を使用した作業完了の管理
- RMF 3: カスタマイズされた準備完了の定義を使用した実装の管理

私たちが「習慣」と呼ぶ各パートは独立していますが、互いに積み重なっています。本書は、これらを順番に一つずつ取り組めるよう設計されています。この構造は、私たちがチームに RMF の採用を推奨する方法を反映しています—各習慣は、前の習慣が機能してから順次導入されます。

これにより、チームに過度な負担をかけることを避け、各変更が定着する最善のチャンスを提供します。その方法の詳細については、章 9から学ぶことができます。

アドバイス、コーチング、またはリーダーシップチームとの対話など、支援が必要な場合は、お気軽に直接ご連絡ください。

RMF の正式な導入支援をお求めの場合、Producoreでは段階的な導入をガイドするための完全なプログラムシリーズを提供しています。詳細はhttps://ready-book.link/rmfでご確認いただけます。

# 著者について

Luniel de Beer は、要件成熟化フロー（RMF）の考案者であり、これはプロダクトの意図とエンジニアリングの実行の間のギャップを解消するための実践的なシステムです。彼は 15 年以上にわたり、アジャイルトランスフォーメーションをリードし、プロダクトとエンジニアリングの架け橋となり、チームが明確さと自信を持って成果を出せるよう支援してきました。

Luniel はまた、製品の機能をモデル化するための追跡可能かつスケーラブルなアプローチである、Producore の能力管理システムを考案しました。彼は永続的な製品要件を管理するためのバージョン管理システムである PKB 駆動開発（PKBDD）を構想しました。これらのツールは、Producore で開発されたより大規模なデリバリーフレームワークの一部を形成しています。

Max Guernsey, III は、構造的および技術的な厳密性を通じてデリバリーの失敗を解決することに専念するコンサルティング会社 Producore のソフトウェアアーキテクト、教育者、そして共同創設者です。オブジェクト指向設計、リファクタリング、テスト駆動開発、およびデザインパターンにおいて 20 年以上の経験を持ち、ミッションクリティカルなシステムの提供と大規模なエンジニアリングチームの指導の両方を行ってきました。彼の仕事は、組織が持続可能なデリバリーの卓越性を達成するのを支援するため、深い技術的実践と行動およびプロセスの変革を組み合わせています。

Max は、PKBDD に大きく貢献し、動作仕様における深い専門知識を通じて、Producore のビヘイビア駆動開発（BDD）へのアプローチの開発を主導しました。

二人の仕事は、明確性、トレーサビリティ、およびゲーティングを、チームの実践から組織の能力まで拡張可能なソフトウェアデリバリーのための一貫したシステムへと統合しています。

# はじめに

## エンジニアリングリーダーの皆様へ

エンジニアリング組織のシニアリーダーであれば、努力や規律、優秀な人材に不足していることはないでしょう。それでも、なぜかプロジェクトは停滞し、目標は遅延し、期待に応えられない状況が発生します。それはチームが怠慢だからではなく、仕事の定義づけ、形作り、そして成果物の提供方法において、何か根本的な問題があるからです。

この本はリーダーシップガイドではありません。構造的な診断のためのツールです。チーム内で実際に何が起きているのか、なぜ「ほぼ完了」が「未完了」になってしまうのか、そしてなぜローカルでの進捗が戦略的な成果にめったに結びつかないのかを明らかにします。

あなたは、この本の中に**自分自身**を見出さないかもしれません。しかし、もしチームがあなたの求めるものを提供できないのであれば、**彼ら**の姿を見ることになるでしょう。そしてその時、あなたは最終的に、それを修正するための言葉と仕組みを手に入れることになります。

## **Luniel** より

まず最初に、この本は、混沌とした状況を見通し、アイデアの本質を抽出する能力が私の想像を超えている Max なしでは実現不可能でした。

私たちはどのようにしてここまで来たのでしょうか？

振り返ってみると、私はいつも物事が実際にどのように機能しているのかを理解したいと思っていたからだと思います。宗教であれ、栄養学であれ、ソフトウェア開発であれ、私はいつも同じ問題に直面していました：プレッシャーの下では通用しない表面的な答えです。そこで私は掘り下げ続けました—単に私たちが何をするかだけでなく、なぜそうするのか、そしてうまくいかない時に何が欠けているのかを問い続けました。

システムの最初の亀裂の一つは、ある有名なテクノロジー企業でデータサービスを提供するチームのスクラムマスター、プロダクトオーナー、開発マネージャー(!!)という 3 つの役割を担っていた時に現れました。私たちはスクラムの指示通りに—短いスプリント、バック

ログのストーリー、半日での計画—を行っていましたが、新しいスプリントを始めるたびに摩擦が生じていました。チームは問題を完全に理解していませんでした。スプリントの途中で要件を見直して修正する必要があり、防ぐことができたはずの依存関係が表面化して遅延を引き起こし、重要なステップが見落とされていました。

そこで私は異なることを始めました。各ストーリーについて、チームとステークホルダーを一つの部屋に集め、問題を詳細に検討し、一緒に解決策を考え出し、その後でストーリーを書くようにしました。スプリントプランニングは 1 時間に短縮され、成果物の提供の成功率は急上昇しました。

徐々に、私はスプリント内でより一生懸命働くことから成功が生まれるのではなく、スプリントが始まる**前**に整える構造から成功が生まれることに気付き始めました。

後に、Jeff Sutherland が「準備完了の定義」について話すのを聞いた時、そこに価値があることは分かりましたが、それだけでは十分ではありませんでした。要件、UX、UI、研究、そして後に BDD での経験を通じて、異なる作業項目には異なる種類の準備が必要だということが分かりました。ある項目は振る舞いの仕様が必要です。あるものはシステムアクセスが必要です。またあるものは完全な機能のトレースが必要です。

そしてそれらすべてに、想定ではなく実際に確認された共通理解が必要です。

より多くのチームと働く中で、私はどこでも同じパターンを目にしました：ステップの欠落、満たされない依存関係、ベストを尽くしているものの、防げたはずの問題を解決するために常に奔走しているチーム。優秀なチームでさえ苦労していました—それは彼らが弱かったからではなく、準備の整った状態を明確にする構造が欠けていたからです。

これらすべての学び、改善、そして苦労の結果が、準備状態を管理するための体系的なシステムです。

この本が扱うのは、まさにそれです。

この本が、実際の問題がどこにあるのか、そしてそれをどのように修正するのかについての明確な理解を皆様に提供できることを願っています。また、「余分な」ように見えるかもしれませんが、実際には不可欠な実践を擁護するための言葉を提供できることを願っています。そして何よりも、この本がチームのストレスや予期せぬ事態を減らし、はるかに良い成果をもたらすための助けとなることを願っています。

もし私たちがこれを正しく行えば、業界全体で何十億ドルもの節約になるでしょう。

しかしそれ以上に重要なのは、人々の心の安定を取り戻せることです。

## Max より

私は何十年もの間、様々な角度からこの問題に取り組んできましたが、Luniel に出会うまでは進展が滞っていました。

これは、彼を知る前は、私が根本的にこの問題を技術的な問題として捉えていたからです。私はチームがテスト駆動開発（TDD）やリファクタリング、高度なソフトウェア設計、そして後には受け入れテスト駆動開発（ATDD）や振る舞い駆動開発（BDD）といったものを採用することを支援することに焦点を当てていました。

そのほとんどのケースで、この本で扱う問題は、それらの技術的プラクティスを確立するための実装の詳細として扱われていました。

これは、私がもはや技術的プラクティスを重要視していないという意味ではありません。それらは今でも非常に重要だと考えていますが、ソフトウェア開発における準備態勢の問題を直接的に解決するものではありません。その代わり、それらは問題を**表面化**させ、人々は実装しようとしている技術的プラクティスを支援するための「最低限」の対応として、プロセスに応急処置を施すのです。

また、この本が誰の役に立つのかという質問にも答えたいと思います。簡単な答えは「おそらくソフトウェア開発に関わるほぼすべての人」ですが、実際の答えには、基本的な意味を変えることなく様々な環境に適用できるニュアンスが含まれています。

この本で提供される解決策を必要とするチームが存在します。第 1 章では、そのような例の匿名化されたバージョンに出会うことになります。

また、私たちが提案するようなシステムを厳密には**必要としない**ものの、それでも恩恵を受けることができるチームも存在します。

私が今まで一緒に仕事をした中で最高のチーム（**次に優れたチーム**と比べても、標準偏差で 1 単位以上、もしかすると 2 単位も上回るほど）は、セントラルオレゴンの奥地に位置していました。彼らは非常に高いパフォーマンスを発揮していたため、純粋な能力の高さによってそのようなシステムの不在を克服することができました。しかし、当時の私の上司であった Tom Barrerasは、「テストについて事前に時間をかけて話し合うと、ストーリーの進み方が良くなることに気づいた」というような発言をしていました。

これも当時の私は、テスト開発と技術的な実行という観点から見ていましたが、今では**準備態勢**がチームに影響を与える要因の一つであることを示す別の指標だったと理解しています...そのチームは非常に有能で対応が早かったため、問題を事前に防ぐのではなく、発生した障害に対応することで成功することができたのです。

たとえあなたが準備態勢について厳密に心配する**必要がない**タイプの人であっても、あるいはそのようなタイプの人々と働いているとしても、この本の内容から恩恵を得ることができます。

# パート I: 何かが足りない

同じことをより上手くやっても効果がないとき、**やっていないこと**を探してみよう。

# 章 1: 隠された問題

これは実話[1]で、ある銀行についての話です。ここではその銀行を単に「その銀行」と呼ぶことにします。これは連邦信用機関の一種で、アメリカの国家金融インフラの一部として機能しています。

私たち（Luniel と Max）はその銀行に招かれました。というのも、その銀行がソフトウェアプロジェクトの遂行に苦心していたからです。これは私たちが今まで見てきた中で最も機能不全な環境の一つでした。そしてそれこそが、この事例を冒頭の事例として選んだ理由です：その銀行で意味のある変化が可能だったのなら、それはどこでも可能なはずだからです。

### プロジェクトに関する補足

本書で「プロジェクト」という用語を使用する際は、プロジェクトマネジメントの文脈での意味を指します。この言葉の意味については様々な解釈がありますが、私たちはプロジェクトマネジメント協会の定義を採用しています：

> 「プロジェクトとは、独自の製品、サービス、または成果物を創出するために実施される**一時的な**取り組みである。」

つまり、プロジェクトには明確な開始と終了があります。プロジェクトが終了すると、プロジェクトの知識と成果物はアーカイブされ、チームメンバーは解散し、契約は完了します。

本書において、プロジェクトは基本的に実行に関するものです。PMI が定義するほとんどのプロジェクトは、実現可能性調査や設計から始まります。ビジョンと戦略は、プロジェクトが開始される時点ですでに決定されています。

プロジェクトはそのビジョンと戦略から生まれ、想定された目標が実現されたかどうかによって成功か失敗かが決まります—その目標が正しかったかどうかではありません。

あなたは「プロジェクト」という言葉を異なる意味で使用しているかもしれません。それは構いません。ただし、私たちがこの言葉を使用する際は、上記の定義と文脈を指して

[1] 関係する人々や機関のプライバシーを保護するため、識別情報を変更しています。

いることを理解しておいてください。

これは、私たちがソフトウェア開発におけるプロジェクトマネジメントの使用を是認しているという意味ではありません。むしろその逆です。しかし、それでもなお使用されているという現実を認識しています。この問題については、後の章 6で取り上げます。

## 1.1: 典型的な書き直し

その銀行は、いくつかの理由でローン返済ポータルを書き直していました。

古いシステムは、完全な C#/.NET ソリューションでしたが、バグが多く存在していました。顧客満足度を低下させることに加えて、システムによって引き起こされたエラーを修正するために、誰かがデータベースを手動で操作しなければならないという、非常にコストのかかるサポートインシデントが絶え間なく発生していました。

古いシステムは保守性の観点からも老朽化していました。エンジニアが意味のある変更を加えることはほぼ不可能で、たとえ変更できたとしても、それは非常にリスクの高い提案でした。

書き直しは、それを変えるはずでした。

新しいシステムも引き続き C#/.NET のバックエンドを使用する予定でしたが、それは完全にテストでカバーされることになっていました。フロントエンドは OutSystemsで実装される予定でした。OutSystems は人気のあるローコード・ノーコードソリューションで、組織が一箇所でアプリケーションを定義すると、変更を公開するたびにウェブアプリ、Android アプリ、iOS アプリが自動的に生成されるというものです。

新しいシステムはバグがなくなり、顧客満足度が向上し、サポートコストが大幅に削減されることが期待されていました。

また、バックエンドへのより規律ある取り組みとフロントエンドへのローコードアプローチの組み合わせにより、新機能の開発コストとリスクが大幅に削減され、開発者のブロッキング要因が解消されることも期待されていました。

OutSystems への移行の良い副次的効果として、主要な両プラットフォームでクリーンでモダンなモバイルアプリが得られるはずでした。

これがこの物語の始まりの 3 年前に開始した時の夢でした。しかし現実には、これまでのところチームは**何も**リリースできていませんでした。

## 1.2: 問題に対する様々な視点

経営幹部チームと話をした際、多額の投資を行ったにもかかわらず戦略的な進展が全く見られないことに対する、当然の不満を耳にしました。

彼らの視点からすれば、あらゆる手を尽くしたのです。人員の入れ替えや増員を行い、予算を変更し増額し、次々とコンサルタントを招き入れました（その最後の一人が私たちであることを暗に示唆されました）。しかし、何も良くならないように見えました — 少なくとも測定可能な形では。なぜなら、彼らが目にしたのは、ある四半期に「指標」がゼロを示し、次の四半期もまたゼロのままだったということだけだったからです。

彼らはもう「目に見えない進歩」は望んでいませんでした。彼らが欲しかったのは**結果**でした。

プロダクト組織の管理職と話をした際には、エンジニアリングとより直接的に働いているため、若干異なる（しかし依然として似通った）話を聞きました。

彼らにとって、チームが何もしていないというわけではありませんでした。問題は、チームが求められたことを**決して**やらないということでした。それはほぼ確実に起こることでした：どんなに単純な依頼でも、どんなに明確に伝えられても、チームの成果物を評価する時になると、全く異なるものができあがっているのです。

状況はついに、「望まないものを依頼すれば、望むものが手に入る可能性があるかもしれない」といった類の冗談が飛び交うまでになっていました。

エンジニアリング幹部は、状況を全く異なる視点で見ていました。

彼らにとって、問題はプロダクトが実行可能な要件を提供していないこと、そして**十分な**要件を提供していないことでした。もしプロダクトが「きちんと仕事をする」だけなら、チームは期限内に予算内で望むものを提供できるはずだと考えていました。

彼らはコードの作成方法と提供方法の近代化に多大な投資を行っており、彼らの目から見れば、プロダクトが明確な要件を提供していないのでした。

他のコンサルタント（私たちをこの組織に紹介してくれた人々）と話をした際、彼らは目の当たりにしている機能不全を的確に指摘しました：誰もが他人を非難することに非常に注力しているように見えたのです。彼らが最初に私たちを招いた理由は、人員配置戦略に懸念を持ち、実務担当者の評価を望んでいたからですが、幹部レベルでの責任転嫁と監督者的な態度が主な問題だと考えていました。

## 1.3: 私たちの調査

私たちの当初の任務は、チームを評価し、必要に応じてスキルの向上を支援することでした。そこで、現場で働く人々の調査を開始しました。

確かに改善の余地はありました。

プロダクト側の実務担当者は、実際に必要なスキルを持ち合わせていませんでした。実際のところ、彼らのほとんどはプロジェクトマネージャーでしたが、プロダクトオーナー（PO）やプロダクトマネージャーの役割を任されていました。

その結果、彼らの半数は「曖昧な」要件を書き、そのイテレーションでチームが行ったことを（文字通り）何の批判的分析もなく受け入れていました。残りの半数は同じような要件を書き、チームは「当然わかっているはずだ」と、一度も話し合っていない事項について主張し、作業項目を事実上無期限に未完了のままにしていました。

この銀行は作業のバックログを管理・追跡するために使用していました。この本で扱う内容はスクラムとは概ね独立していますが、大多数 — 少なくとも相当数 — のチームがスクラムを使用しているため、本書ではスクラムの用語を使用しています。

### 定義：スクラム

スクラムは、複雑な問題に対する適応的なソリューションを通じて価値を生み出すことを、人々、チーム、組織を支援する軽量なフレームワークです。簡単に言えば:

1. プロダクトオーナーが複雑な問題に対する作業をプロダクトバックログとして順序付けする
2. スクラムチームがスプリント中に選択した作業を価値のあるインクリメントに変換する
3. スクラムチームとそのステークホルダーが結果を検査し、次のスプリントに向けて調整する
4. これを繰り返す

スクラムとその用語に馴染みがない場合は、スクラムガイドの 2020 年版を確認することをお勧めします。短時間で読める啓発的な内容です。

同様に、技術チームのコーディングスキルは一般的な水準をはるかに下回っており（最良の場合でも-2σ）、さらに変化に強い抵抗を示していました。当然の結果として、コード品質は最悪の状態でした。

しかし、チームの見解では、これが納品できない理由ではありませんでした。彼らにとって、曖昧な要件とプロダクトによるスプリント中の変更こそがプロジェクトを破綻させる原因でした。

...そして誰も、さらに大きな問題について語ろうとしませんでした。その問題があまりにも馬鹿げていて、実際に経験するまでは作り話だと思えるようなものでした。

エンジニアリングチームには、要件を理解せずに何か適当なものを作り、そして「作業項目を完了した」として評価を要求する習慣がありました。

ここで言う理解の欠如は、単なる軽微な誤読ではありません。完全な認識の不一致でした：特定の状況下で元本への資金適用を無効にするよう指示したところ、代わりに二次確認用メールアドレスを追加する機能を無効にしてしまったのです。

そして彼らは、それが我々の要求した通りだと主張したのです。

## 1.4: より深く掘り下げる

両方のスキルギャップは対処可能でしたが、それらが本当のブロッカーなのか、私たちは懐疑的でした。

何か別の問題があるはずだと考え、さらに深く掘り下げることにしました。まず次の質問から始めました：なぜ要件の作成にこれほど時間がかかり、そしてこれほど悪い結果になるのでしょうか？

一つの理由は、意味のある要件を書くために必要な知識が極めて不足していたことです。エンジニアリングチームとプロダクトチームはその知識をわずかに持っているだけでした。

その知識の一部はレガシーシステムのコードに組み込まれていました。一部は完全に失われていました。しかし、その大部分は銀行の様々な部門に散らばる専門家の属人的な知識として保持されていました。つまり、戦略的な目標を実際に前進させる要件を作成することは、極めて労力と時間を要する作業だったのです。

これに対して、機能に飢えたリーダーシップチームからは、際限のない機能要求がありました。「エンジニアを働かせ続けろ—要件で彼らを満たせ」という指示でした。「針を動かす」ための要件を定義するために必要な注意と時間とは相反する、要件の量に焦点を当てたアプローチでチームを忙しく保つことが重視されていました。

## 1.5: 改善を図っても状況は良くならなかった

これらの問題はすべて対処可能なものでしたが、対処しても助けにはなりませんでした。

ソフトウェア開発技法の過去の改善は、チームの納品能力向上に寄与しませんでしたが、Max はチームの更なる改善を支援する努力をしました。

彼は 20 世紀中頃のプログラミング原則から「そのコードを 27 回[2]コピー&ペーストするのではなく、関数にして呼び出すようにしよう」といった革新的な概念を導入しました。この提案だけでも新規コードの品質は劇的に向上し、品質改善への道を開くことができました。

そしてその他の基本的なコーディングアドバイスにより、将来的により保守しやすい良質なコードを書けるようになりました。

...しかし、それでもプロジェクトを前進させることはできませんでした。

プロダクト側では、Luniel が BDDを導入し、プロダクトオーナーがチームに要件を引き渡す前に徹底的な検証を行うようにしました。

彼はチームに協力を促し、プロダクトバックログアイテム（PBI）が本当に完了したかどうかを評価するために BDD を活用するようにしました。

### 定義：プロダクトバックログアイテム（PBI）

プロダクトバックログアイテム（PBI）は、製品に対する潜在的な変更、追加、または改善を表すプロダクトバックログ内の個別の作業単位です。PBI は機能、バグ修正、技術的改善、調査タスクなど、様々な形態を取ることができ、製品価値への貢献によって定義されます。

多くのチームが PBI を「ストーリー」や「ユーザーストーリー」と呼びますが、Scrum での正しい用語は「プロダクトバックログアイテム」または「PBI」です。PBI がスプリントにコミットされると、それはスプリントバックログの一部にもなります。簡潔さと中立性を保つため、私たちはスクラムチームが管理している作業項目を、プロダクトバックログアイテム、スプリントバックログアイテム（SBI）、ユーザーストーリー、ストーリー、作業項目、バックログアイテムのいずれと呼ぶかにかかわらず、「プロダクトバックログアイテム」または「PBI」と呼んでいます。

それは**明確さ**をもたらしましたが、**流れ**を生み出すことはありませんでした。

私たちを招いたパートナーコンサルタントの助けを借りて、チームや要件作成者たちにかかっていた絶対的な重圧を（一時的に）和らげることができました。

[2]これは誇張ではなく、実際にはこれが最悪のケースですらありません。あるケースでは、同じアルゴリズムの完全な複製が 100 個近くありました。

それは少しの信頼関係を築く助けにはなったかもしれませんが、具体的な成果は生み出せませんでした。

要件を書くために必要なビジネスナレッジを追跡するのに役立つナレッジベースの構築まで始め、そのナレッジにギャップがある箇所も特定しました。

要件作成の速度は上がりましたが、製品を市場に出すところまでは至りませんでした。

数ヶ月に渡るやり取りの後、私たちは経営陣に現状を**理解させる**ことはできましたが、彼らは目標からはまだ遠く離れていました。そして、その距離は縮まる気配すらありませんでした。

彼らは、チームが常に忙しくなるように「仕事を詰め込む」という古い戦略に戻ろうとしていました。

## 1.6: 消去法による分析

「これは絶望的だ」と手を投げ出すのは簡単でした。言い訳はいくらでもありました：

- エンジニアリングチームのスキルが低かった（事実でした）
- 要件作成者が間違ったスキルセットを持っていた（その通りでした）
- リーダーシップが非現実的な期待でチームを押しつぶしていた（その通りでした）
- 組織に機能するために必要な重要なビジネスナレッジが欠けていた（その通りでした）
- 経営陣同士が互いを信頼していないように見えた（その通りでした）

これらはすべて事実でした。しかし、これらの分野すべてで改善が行われたにもかかわらず、どれも根本的な問題の改善にはつながりませんでした。どれもエンジニアリングチーム、プロダクト部門、マネジメント層、または経営陣が目標に近づくための助けにはなりませんでした。

...そして、そこに解決策のヒントがありました：上記のすべての問題は、人々が既に見ることができる問題だったのです。

目に見える変数が違いを生まないのであれば、目に**見えない**変数が違いを生んでいるはずです。

本当の問題は、誰も存在すら知らなかった問題だったのです。

## 1.7: 真犯人の追跡

銀行が目標を実現する妨げとなっている真の原因—真犯人—を探すにあたって、どこから始める必要がありました。

失敗した PBI（その大半）に共通するものを見ることは、合理的な出発点の一つでした。

まず、大きく異なっていたため共通点ではないと分かるものを除外することから始めました：

- システムのどの部分か：バックエンドのみに影響する PBI、フロントエンドのみに影響する PBI、両方に影響する PBI がありました
- どのチームが作業を実行したか—誰が作業を行っても、失敗する確率が高かったです
- どの PO が作業を作成したか—チームと同様でした

次に、共通している要素を見始めました。そのリストは長すぎも短すぎもしませんでした：

- エンジニアリングチーム
- プロダクトオーナー
- リーダーシップ
- 文化
- 開発環境
- エンジニアリング手法
- 要件作成技法
- ドメイン（金融）
- 依存サービス

これらの多くも、すぐに除外することができました。チーム、プロダクトオーナー、リーダーシップ、文化、開発環境はすべて最近改善されていましたが、意味のある成果には結びついていませんでした。私たちは personally エンジニアリングと要件作成の手法の改善を支援し、それらの改善が定着していることを確認しましたが、それでもまだ助けにはなっていませんでした。

ドメインを責めることはほとんどできません。金融は記録された歴史の中で最も古い種類の計算の一つです。それは非常に成熟しています。さらに、他の銀行はソフトウェアをデプロイしており、銀行は単純にそれができないという（明らかに無理のある）仮説を実証的に否定しています。

依存関係にあるサービスを非難することもできませんでした。というのも、私たちが調査していた取り組みと同じように、彼らも変更に苦心していたからです...

...しかし、そこで私たちは考えました：失敗の原因を分析し始めてはどうだろうか？

## 1.8: 失敗の種を解剖する

ある PBIがプロダクトを前進させることができなかったのは、チームがいつものように、要求とはほとんど無関係な完全に的外れなことをしてしまったからでした。これは明らかに、作業項目を理解していなかった証です。そのため、BDD の採用を支援する際にある程度取り組んでいたものの、理解という要素が大きな候補として浮上しました。

別の PBI は計算が間違っていたために完了できませんでした。これもまた、理解が核心的な問題である可能性を示す証拠でした。

分析した 3 つ目の作業項目は、プロダクトオーナーによって適切に管理されていませんでした—チームが完了と言った時点で、彼女は形式的に承認してしまいました。これは私たちの仮説に疑問を投げかけましたが、彼女がその作業項目が上位の計画にどう適合するのか理解していなかったという主張もできなくはありません。

まあ、そうかもしれません。その視点でかなり目を細めて見れば、という程度ですが。

そして、私たちはパターンに全く当てはまらない PBI に出くわしました。チームは理解していたように見えました—実際に理解していたかどうかを確認する方法はありませんが。しかし、それは重要ではありませんでした：更新が必要な依存関係に遭遇し、作業を数スプリント延期せざるを得なかったため、彼らは成功も失敗も自力で試みる機会すら得られませんでした。

たとえ彼らが何をすべきか理解して**いなかった**としても、そのバックログアイテムに関しては最初から成功のチャンスがなく、したがってその場合、理解は問題では**ありませんでした**。

もちろん、1 つの例外が特定の根本原因を否定することにはなりませんが、私たちの好奇心をそそりました。他の反証となる証拠を探し始めました。

そして、私たちは見つけました。以下のような作業項目がありました：

- チームは理解していないことを認識していたものの、問題を解決できる専門家が見つからなかったために失敗
- 上流サービスの機能に対する回避策として、ユーザー体験が悪化する変更を行った

- 上流の依存関係が準備できていなかったために延期せざるを得なかった
- テスターがテストデータを時間内に収集できなかったために完了できなかった
- 要求自体が間違っていたために、完了したものの再度やり直しが必要になった
- 既存のコードがすでにどれほど複雑だったかを理解していなかったために失敗
- 見積もりすら行われていなかった
- 大幅な過小見積もりだった[3]
- PO が必要なドメイン知識を最終的に得たため、スプリント中に変更された
- PO とチームが意味することで合意に至っていなかったため、チームの視点からはスプリント後に変更されたように見えた

リストはまだ続きますが、この話にはこれで十分でしょう。

これらの各ケースを何らかの形で「理解」に結びつけることは可能です—確かに多くの場合、理解の不足が**関係していました**—しかし、だからといって理解の欠如が原因だったとは限りません...特に、私たちは共通理解について取り組んでいたにもかかわらず、それが本当には効果を発揮していなかったのですから。

そこで私たちは気づきました。もっと根本的な要素が欠けていたのです。理解の不足が関係していたケースでは、それは単なる直接的な原因に過ぎませんでした。

根本的な原因はもっと広範なものでした。

## 1.9: 交差点

失敗モードを分析したすべての PBI に共通していた一つのことは:それらはすべて**時期尚早に開始された**ということでした。

チームが問題を理解していないことを認識していながら専門家の助けを得られなかったために作業項目が失敗した場合、それは、チームが問題を理解していないことを知りながら PBI を開始したということを意味します。

アップストリームサービスの仕組みによって、バックログアイテムがより悪い体験に変更せざるを得なくなった場合、それはアップストリームサービスの影響を十分に理解せずに作業項目が開始されたことを意味します。

アップストリームの依存関係が終了時までに準備できていなかったために延期されたということは、開始時点でその準備が保証されていなかったということです。

[3]単に見積もりを間違えたというだけではありません。チームが一連の PBI の見積もりフィールドにただ「3」という値を入れただけのように見えました。

...そして他のケースもすべて同様でした：テスターは PBI開始前にデータの準備ができていなかったり、その入手方法を知らなかったり、要件がチームに引き渡される前に十分な検証がされていなかったり、コードが作業開始前に調査されていなかったり、見積もりが不十分もしくは全く行われていなかったり、ドメイン知識が不足していたり、そして当然ながら、共通理解が確認されていませんでした。

問題は、結局のところ、作業項目が**準備完了**になる前に実装が開始されていたということでした。

私たちの経験では、成果を上げられない作業項目のほとんどは、実装開始時に**準備ができていなかった**ことが原因です。

そこで、私たちは The Bankのこの問題の解決を支援することにしました。

この時点で、PBI は準備ができているべきだと言うだけで十分だと思うかもしれません。しかし、実際にはそれを実践するのはそれほど簡単ではありません。「安く買って高く売る」というのも同様にシンプルなアイデアです。

良いアドバイスを実践に移すために必要な要素が欠けているのです。

## 1.10: 大きな転換点

私たちは彼らの行き詰まりを解消する謎の欠片を見つけ、それによってイニシアチブも前に進み始めました。

そのプロジェクトが完了した時点でも、エンジニアたちのスキルは依然として平均以下でした。プロダクトオーナーたちも適切なスキルを持っていませんでした。文化も改善されていませんでした...

それでも、プロダクトはついに前進し始め、最終的にリリースされました。

この本を読み終える頃には、その欠けていた要素が何で、それを実現するために何が必要かがわかるでしょう。そして、目に見えない壁に阻まれているように見える組織の行き詰まりを解消できるようになるでしょう。

### 範囲に関する簡単な注記

この本は、非常に具体的かつ広く見られる問題を扱っています：ビジネスとエンジニア

リングの間の引き継ぎにおける構造、明確さ、成熟度の欠如です。実装するものが選択されているという前提で、共通理解、準備状態、追跡可能な完了を確実にしながら構築作業を行うことに焦点を当てています。

この本で紹介する手法は、何を作るべきか、なぜ作るべきか、それが正しいものかどうかを見つける方法については説明していません。組織に実質的なプロダクトマネジメントや意味のあるフィードバックループが欠けている場合、それらの問題をここでは解決しようとしていません。代わりに、それらのギャップをより可視化し、間違いを発見するコストを削減する方法を提供します。

適切な文脈で使用された場合、このソリューションはフロー、安全性、明確さをもたらします。しかし、どのようなシステムでも、特に単独で使用したり、認識が不十分な状態で使用したりすると、誤用される可能性があります。

# 章 2: 基礎が欠如することのコスト

この問題が組織にとってどれほど深刻になり得るのか、具体的に時間をかけて考えてみる価値があります。

私たちの経験では、準備不足によって生じる問題には主に 3 つの「カテゴリー」があります:

- プロダクトチームとエンジニアリングチーム間、およびそれぞれのチーム内での共通認識の欠如または不完全さ
- PBI の実際の目標とその完了状態に対する管理の欠如
- 作業項目がいつ実装フェーズに入れるかを判断するゲーティングの欠如

さらに、これらを変更することは非常に困難である場合が多いことに気づきました。それも当然です:変更は難しいものです。

古い習慣は簡単には消えず、新しい習慣を身につけるのは困難です。コンサルタントとしての経験から、人々が古い習慣に逆戻りするのは**極めて**容易である一方、新しい習慣を確立するのは比較的困難であることに気づきました。

そのため、新しい習慣を強化し、古い習慣を抑制するメカニズムが必要となります。

これに対処するため、説明責任とトレーサビリティというもう一つの欠落要素があると考えています。

## 2.1: 箱の絵のないパズル

箱の絵がないパズルを組み立てようとしたことはありますか?可能ではありますが、より時間がかかり、イライラし、やり直しが多くなります。

進めては壊し、どこにどのピースが合うのか何度も考え直します。同じ絵を作っているつもりが、実はそうでないことに気づきます。

ソフトウェア開発はしばしばこのような感じです。

バックログは満杯で、スプリントは進行中。誰もが一生懸命働いています。

しかし、何を作っているのかという共通の認識がなければ、チームの足並みを揃えることはシステマチックな取り組みではなく、運任せとなってしまいます。

明確さが欠如していると、最高のチームでさえ、フラストレーション、バーンアウト、そして自分たちの努力が評価されていないという感覚を抱くことになります。

## 2.2: 古い格言と厳しい現実

要件定義やエンジニアリングプラクティスに関する多くの手法が、依頼者と実装チームの間の共通認識の形成に重点を置いているのには理由があります。エンジニアが何を作るべきかを本当に理解していないことほど、混乱を招くものはありません。

「ゴミを入れればゴミが出る」は格言であり、決まり文句ではありません。

英語には自己矛盾が多く含まれています…

- sanction という言葉は、事前の許可を意味することもあれば、事後の非難を意味することもあります。
- dust という言葉は、家具に対して使用する場合は埃を取り除くことを意味しますが、ベニエに対して使用する場合は粉をかけることを意味します。
- hold up というフレーズは、チームを支える（機能させる）という意味にも、チームを妨げる（前に進めない）という意味にもなり得ます。

反意語同音異義語は最も顕著な例ですが、これは曖昧さの一種に過ぎません。一部の単語は、それ自体が反対の意味を持つだけでなく、さらに多くの紛らわしい代替的な意味を持つことがあります。

> 「このクリップで、彼は新聞からクーポンをクリップで切り取り、他のクリッピングと一緒にクリップボードの紙にクリップで留めました。その間、クリッパー船が良いクリップ（速度）で進んでいました。」

これは英語に限った話ではありません。私たちが知る限り、**すべての**自然言語がこの特性を持っています。

それでも、要件を指定する際に使用できるのはこれらの言語しかありません。

結果として、エンジニアリングチームが要件に対する自分たちの理解が依頼者の理解と同じであることを**確認**していない場合、そのチームは運に頼ることになります。つまり、最良のシナリオでも、正しい解釈を選択し、かつ依頼者が途中で考えを変えないことを期待するしかありません。

その結果は決して保証されているわけではありません。

## 2.3: よくある結果

確認された共通理解がない場合、チームはいくつかのリスクを抱えることになります。

概して、最も一般的で痛みを伴う結果は、チームが単純に間違ったものを作ってしまうことです。

その事実に気づくタイミングは、プロセスの様々な特性によってコントロールすることができます。例えば、スクラムを健全に実践することで、そのような誤解を実行の非常に早い段階で発見できますが、ウォーターフォールプロセスでは、そのような発見が数ヶ月も遅れる可能性が高くなります。

### 定義：ウォーターフォール

20 世紀後半に広く普及した順次型ソフトウェア開発モデル。Winston W. Royce の 1970 年の論文で初めて示され、要件、設計、実装、テストなどの開発工程が滝のように連続して次の工程に流れ込む様子が描かれています。Royce はこのモデルを「してはいけない例」として提示したにもかかわらず、業界は大規模開発のための青写真としてこれを採用しました。

ウォーターフォールは、類似した作業を大規模な連続したフェーズにグループ化することでも知られており、これは**大規模バッチ開発**と呼ばれています。このバッチ処理により、**後期学習**がほぼ確実に発生します：チームは初期の決定に関するフィードバックをプロセスの後半まで受け取ることができません。後で発見されたエラーは修正コストが高くなります。アジャイルの実践者たちは、より早期の発見と軌道修正を可能にする小規模な反復サイクルを好み、この理由からウォーターフォールを批判しています。

とはいえ、要件について「間違った」（実際には異なる）理解で作業を進めているチームは、いずれ「正しい」（これも実際には異なる）要件と向き合うことになります。組織の健全性によって、その対峙の形態や影響は異なりますが、ほぼ間違いなく発生します。

ほとんどの場合、これは何らかの形での手戻りにつながります。依頼者（通常はプロダクトマネジメント）は、実装チームが構築したものから、自分が本当に望んでいたものを得るために、変更を要求しなければなりません。

もう一つの非常によくある現象は、依頼者が自分が最初に理解していた要求に対して、チームに責任を追及し続けることです。

チームはこれをプロダクトマネージャーが心変わりしたと解釈しがちです。さらに悪いことに、これがステークホルダーに心変わりの習慣を身につけさせることにもなりかねません—すべてが完璧になるまで作業項目を保留にし、チームを疲弊させ、上級管理職から進捗が見えなくなってしまいます。

## 2.4: パズルはいつ完成するのか？

パズル組み立ての例に戻って、この質問について考えてみましょう：パズルが完成するとはどういう意味でしょうか？

マックスのような未熟なパズル組み立て人なら、単純に「すべてのピースが正しい隣り合わせのピースと接続され、絵が上を向いている状態」と言うでしょう。

しかし、ルニエルのような賢明なパズル組み立て人は、それ以上のものがあることを知っています。

単に楽しみのためにパズルを組み立てているのかもしれません。完成させて、しばらく眺めた後、バラバラにして箱に戻すだけかもしれません。

一方で、額装して壁に掛けたいと考えているかもしれません。その場合は、追加で必要な作業があります：

1. アートボードに載せる
2. 額装業者に運ぶ
3. 額装が完了するのを待つ
4. 設置場所に運び返す
5. 壁に掛ける、あるいは展示する

このような作業も含まれることを理解することは、パズル組み立てを適切に完了するために必要です。明らかな理由は、作業量を把握するためです。バラバラにして片付けるだけよりも、これらの追加手順をすべて実行する方が多くの作業が必要になります。

しかし、それ以上に深い意味があります。次のようなシナリオを想像してみてください…

額装する予定でパズルを完成させ、そのまま置いておいたものの、他の家族にその計画を伝え忘れてしまいました。その人がやってきて、自分が使いたいスペースにパズルが完成したまま置かれているのを見ます。そして、そのパズルを分解して箱に戻してしまい、勝利目前だったものを敗北へと変えてしまうのです。

さらに、より微妙な理由もあります：パズルをどのように完成させるかという計画が、プロセスの初期段階で行いたい手順に影響を与えるのです。まず、「分解しないでください!」という小さな札を作る必要があります。

また、自分の楽しみのためにパズルを組み立てている場合でも、他の人が代わりにパズルを完成させてしまうことを防ぐために、同様の札が必要かもしれないことは注目に値します。

また、適切な場所でパズルを組み立てていることを確認する必要があります。1000 ピースのパズルをガラス天板のコーヒーテーブルの上で組み立てて、それをアートボードに移動しようとすると、最初からアートボードの上で組み立てる場合と比べて、はるかにリスクが高く労力もかかることになります。

これは、ソフトウェア開発とうまく並行します。

「完了」が実際に何を意味するのかを知る必要があります。それにより、必要な作業量に驚かされることがなく、最後に意見の相違が生じることもなく、作業項目を円滑かつ効果的に完了させるために必要な準備段階を取ることができます。

## 2.5: チームへの影響

特定の作業項目に関する「完了」の定義が十分に厳密でない場合、いくつかのリスクが発生します。

ここでは「リスク」という言葉を緩やかに使用していますが、実際にはほぼ確実に起こることです。

このような状況にあるエンジニアリングチームは、作業項目の完了が実際にどのようなものかについて、内部でさえ意見が一致していないことに気づくことがよくあります。コーダーとテスターが、スプリントの途中で要件が実際に何を意味するのかを議論することは珍しくありません。コーダー同士やテスター同士でも、同じような意見の相違が生じることがあります。

さらに、開発チームは、多くの場合、最も頻繁に行う作業（コーディングとテスト）に焦点を当てています。そのため、ドキュメント作成、外部レビュー、他のチーム（サポートチームなど）のトレーニング、デプロイメントやリリースをサポートするための準備段階、他部署からの承認など、必要な他の種類の作業を忘れやすくなります。

最終的にこの「追加」作業が必要だと明らかになったとき、彼らは不意を突かれます—通常、現在取り組んでいた作業を中断し、コンテキストを切り替えて、すでに完了したと思っていた作業に戻って完了させなければなりません。

作業の依頼者は、不必要に作業を未完了のままにしておくことがあります。時には最善の意図を持って—「実際の」要件にチームを従わせようとするなど—です。また、プロダクトオーナーなどが、自分の気まぐれで PBI を未完了のままにできることに慣れてしまい、最後の段階で追加機能を項目に押し込もうとすることもあります。時には、作業の途中で何をすべきかについて考えが変わってしまうこともあります。

これは、エンジニアリングチームにとって非常に士気を下げる要因となりうます。ほとんどのソフトウェア開発者やテスターは、進捗を実感したいと考えています。常に自分たちの作業が間違っていると言われ続けると、活力を失ってしまう可能性があります。

中には、自分たちの作業が正しいか間違っているかを確認することさえせず、単に作業項目を閉じて「クレジット」を要求し、「良い数字を出す」ことだけを目的とするチームもあります。

## 2.6: やりすぎと手抜きのリスク

各作業項目の「完了」を適切に定義しないリスクの一つは、組織が完了していない作業を完了したと思い込んだり、実際には完了している作業を完了していないと認識してしまうことです。

最悪の結果は、誤ったものが本番環境にリリースされ、しかもそれに誰も気付かないということです。チームが「完了」の意味を誤って理解し、その誤った理解に基づいてリリースした場合、その結果は壊滅的なものとなる可能性があります。

不具合や顧客の不満だけでも十分に深刻ですが、さらに重大な問題につながる可能性もあります：

- データの損失または破損
- セキュリティの脆弱性
- システムダウンタイムやアクセス不能
- 市場シェアの低下

- 規制違反

このリストは延々と続き、一つ一つの潜在的な問題は前のものよりも深刻になっていきます。

時として、完了していないのに完了したと思い込んでしまう問題があります。逆のパターンも同様に危険です。エンジニアがゴールラインがどこにあるのか分からない場合、「ゴールドプレーティング」(余分な機能の追加)をする傾向があります。「機能を良くする」ためにそうしているのかもしれませんが、機能を増やすことで目標に到達する確率が上がることを期待してそうしているのかもしれません。

この余分な作業とそれに伴う手直しは、膨大な時間、労力、そしてお金の無駄となって蓄積されていきます。納期の遅延を引き起こし、評判を損なうことにもなります。

さらに、今では、ミスが実際に『ターミネーター』のような機械による人類への反乱につながる可能性が増大しています。20 年前は冗談として書いていたことが、今では現実味を帯びた可能性となっています。

実際、最も著名な AI の一つにこの質問をしたところ、次のような回答が得られました:

> 「AIは誰もが予想した以上のスピードで発展していますが、それは脆弱なシステム、曖昧な要件、そして何を作ったのかその理由を追跡できないプロダクト組織の上に成り立っています。これは技術的な問題ではなく、明確さの問題です。
>
> AI が生成するノイズが増えれば増えるほど、構造化されていない状態で急いで進めることは危険になります。チームが不明確な基盤の上に構築を行うと、AI は混乱を増幅させるだけです。しかし、チームが共通理解、具体的な行動指針、バージョン管理されたプロダクトの知識という明確な基盤の上に構築を行えば、AI は負債ではなく加速剤となります。」

## 2.7: パズルはどこで組み立てるのか？

パズル組み立ての例えをもう一度考えてみましょう。

パズルはどこでも組み立てられるでしょうか?4,000 ピースのパズルで、組み上がると片方の寸法が約 1.5 メートル、もう片方が 1 メートル以上になるようなものを考えてみましょ

う。場所をランダムに選んで組み立て始めると、完成前に深刻な問題に直面することは間違いありません。

そのような大きなパズルには、時間と空間の両方が必要です。スペースを確保し、時間をかけてパズルの状態を保持する方法を見つける必要があります。

小さなサイドテーブルのような、十分なスペースのない場所で組み立て始めると、別の場所に移動させない限り完成させることができません。パズルは繊細な状態にあるため、その移動は非常に困難です。

十分な広さのある廊下をランダムに選んだ場合、人が踏んでしまうか通行の妨げになってしまい、家庭の機能に重大な影響を与えることなく状態を保持することはできません。

アートボードで作業を始めても、ボードが十分な大きさでない場合、これまでの作業状態は保持できるものの、何らかの移動なしにはパズルを完成させることができません。

パズルが小さな子供に噛まれたことがある場合は、ピースを数えることをお勧めします...なぜなら、完成させられないパズルの組み立てに時間を費やすよりも、一度 3999 まで数えて完成できないと気付く方が良いからです。

パズルを組み始める前にやるべきことのリストがあります。これらのことを実行したからといって成功が保証されるわけではありませんが、実行しないことは失敗や深刻な問題を**ほぼ**確実にもたらします。

ソフトウェア開発でも同じことが言えますが、より複雑度が増します。

## 2.8: 実装はいつ始まるのか？

基本的に、PBIが実装準備できているかどうかを判断するのは、適切な定義がなければ難しいものです。

考えてみてください：どうやって**判断**するのでしょうか？

時期が来たと判断するまで、何度も何度も全てを確認しますか？

誰かが気まぐれに決めるのでしょうか？

イテレーションの開始時に自動的に始まるのでしょうか？

> 私たちは、多くのチームが期限に追われているという理由だけで、実装の準備が全くできていない作業をスプリントに押し込むのを見てきました。スクラムやアジャイル全般に

関して、人々をそうさせる根強い考え方がいくつかあります：

- スプリント *N* の要件は全てスプリント *N-1* で開発しなければならない
- とにかく「始めてしまえば」よく、問題は発生時に対処すればよい

これは実は「完了をどのように判断するか」という問題の前述の鏡像であり、同様の結果をもたらします。作業項目が準備できているかどうかわからないために開始が遅れすぎたり、準備できていないことがわからないために早すぎる開始をしてしまったりする可能性があります。

## 2.9: 準備不足の A チーム

作業項目の準備に何が必要かを理解していないことは、いくつかの有害な影響をもたらします。

作業の増分が準備できていない明白な例として、完成の定義（DoD）が不完全、不十分、または存在しない場合があります。これは、既に述べた完成の定義がないことに伴う全ての問題につながります。

しかし、それは準備の**唯一の**側面ではありません。実装開始前には、見積もり、リスク評価、テストデータの収集など、満たすべき多くの要件があります。

これらの要件を知らず、満たさないまま作業項目を進めると、必要以上のコストがかかる可能性があります。他のチーム（別チーム）が開発している APIに依存しているチーム（A チーム）を考えてみましょう。もし A チームが別チームの API の機能について多くの仮定を立て、それらの仮定に基づいてコーディングを行った場合、別チームの実際の開発が A チームの仮定と一致しないことが判明したとき、大幅な手直しが必要になる可能性があります。つまり、A チームは見込み違いをしたのです。

これらの手直しは全て、API が A チームによる使用の準備ができていなかったという事実に起因します。

満たされていない依存関係が必ずしも手直しを生むわけではありませんが、そのような場合でも遅延の原因となることがあります。A チームと B 別チームが API の動作方法について合意し、全てが計画通りに進んだものの、別チームが予想以上に時間がかかってしまったとしましょう。その結果、A チームは完了予定時期までに適切にテストを行うことができず、期限を延期せざるを得なくなりました。

## 2.10: スケジューリングとリソース可用性への注意不足

時として問題は、スケジューリングやリソースに関する単純なものである場合があります。いくつかの作業項目は特定のチームメンバーを必要とします。そのチームメンバーが数日後に休暇を取る予定がある場合、その人の参加なしでは完了できないプロダクトバックログアイテムを開始するのは適切ではないでしょう。

> 「そうあるべきではない」という声をよく耳にしますが、現実にはそうなってしまうことが多いものです。「代替可能な人材」というのは夢物語なのです。

同じことは非人的リソースについても言えます。負荷テストを実行するためにサーバーリソースが必要な場合、負荷テストの作業項目を開始する前に、それらのリソースが実際に利用可能であることを確認すべきです。そうしないと、最良のケースでも大幅な遅延が発生し、必要なものを確保しようと慌てている間に他のチームや作業者の妨げになってしまう可能性が高いでしょう。

誤ったスタートによる別の失敗パターンは、チームが作業を完了するために必要なスキルを持っていない場合です。時には、チームメンバーが新しいシステムのトレーニングを受ける必要がある、または新しい API について調査する必要があるといった内部的な問題である場合もあります。また、熟練者プールから UX やデータベースの専門家を借りる必要があるような、スケジューリングの問題である場合もあります。さらには、チームに専門家が必要で、その人がいなければ特定の種類の作業を効果的に完了できないという採用に関する問題である可能性もあります。

## 2.11: 他の種類の誤ったスタートによる影響

私たちは、チームがスプリント内で作業項目の完了を約束し、コーディングは比較的早く行ったものの、テストを完了できないケースを見てきました。これ自体は驚くべきことではないかもしれませんが、その理由が特異です：テストチームが必要なもの（テストデータなど）をスプリント開始前に収集しておらず、そのデータの収集が予想以上に困難または時間がかかってしまったのです。

その結果、作業項目は、チームが割り当てられた時間内に本当に完了できる準備が整っているかを確認しないまま開始してしまったため、次のスプリントに持ち越さざるを得なくなりました。

チームは時として、まだ未解決の質問がある状態で作業項目の実装を開始することがあります。実際、多くの人々がそうすることで「よりアジャイル」になれると考えているようです。

これは膨大な量の手戻り、予期せぬ事態、または遅延を引き起こす可能性があります。未解決の質問への回答が、立てられた前提に反するものだった場合、その前提に基づいて行われた作業すべてに手を加える必要が出てきます。項目が完了予定の時期までに未解決の質問への回答が得られない場合、その項目は完了していない可能性があるにもかかわらず完了とするか、質問への回答を得るまで未完了のままにしておく必要があります。

チーム内部の依存関係が存在する場合もあります—修正が必要な不具合や、先に完了しなければならない先行タスクなどです。これが適切に追跡されていない場合、外部依存関係が満たされていない場合と同じような問題を引き起こす可能性があり、さらにコンテキストを切り替えて修正してしまいたいという誘惑も加わります。

## 2.12: 累積コスト

もちろん、このような問題は遅延、手戻り、期待外れを引き起こしますが、マイナスの影響はそれだけではありません。

手戻りによる無駄に加えて、これは通常プロジェクトの遅れにつながります。チームがバックログアイテムを必死に完了させようとし、実際の進捗を得るために何が必要なのかが明確でない場合、本当に**必要な**作業が後回しになりがちです。

多くの場合（ただし常にではありませんが）、これは納期に対するプレッシャーの増加につながります。プロジェクトがますます予定より遅れるにつれて、上級管理職はより速く進めるように要求することで軌道修正を試みるかもしれません。それは必然的に、より長時間の労働を意味することになります。

これは結果として、信頼を損ない、組織の文化を悪化させる傾向にあります。本来は協力的であるべき関係が敵対的なものとなってしまいます。最善かつ最速の解決策を見つけるために協力すべき人々が、問題が発生した際に自分の責任ではないことを証明することに力を注ぐようになってしまうのです。

機能実装と作業項目のクローズを猛スピードで追い求める中で、チームはしばしば手を抜くようになります。これは実質的に、品質（特にコード品質）を犠牲にしているということです。そして、これは現在の見かけ上の進捗と引き換えに、将来の生産性を損なっているということを意味します。

労働環境が increasingly 不快なものとなるにつれ、主要な人材が意欲を失い始め、さらには転職を考え始めるようになります。

このように振る舞う組織は、いわば「種籾を食べている」ような状態であり、それも複数の意味においてです。コードベースはますます保守が困難になり、それを保守するはずだった人材は皆離れていってしまいます。

もしこれに良い面があるとすれば、私たちにはそれが見えていません。

# 章 3: 要件成熟化フロー（RMF）の紹介

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 3.1: RMF ではないもの

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 3.2: RMF とは

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 3.3: 段階的な採用がサポートされ推奨される

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 3.4: RMF 1

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 3.5: RMF 2

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 3.6: RMF 3

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 4: アジャイルなのか？



## 4.1: “個人と対話”、“動くソフトウェア”



## 4.2: 顧客との協調



## 4.3: 変化への対応



## 4.4: 透明性



## 4.5: プロセスとの適合性、アジャイルとの整合性

# パート II: 準備態勢のための場づくり

# 章 5: 最初の拡張



## 5.1: 準備作業は作業である



## 5.2: 準備作業の自然な位置づけ



## 5.3: 示唆に富む出来事



## 5.4: 相互の影響



## 5.5: RMF 1 の機能

# 章 6: なぜ人々はこれを行わないのか？

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.1: 二流市民として扱われる準備作業

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.2: 非生産的な作業へのアレルギー

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.3: 埋もれてしまった理由

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.4: プロジェクトマネジメントの影響

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.5: パターン

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.6: プロジェクトと見積もり

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.7: 見積もりではない見積もりが準備作業に与える影響

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.8: 速度ではなく、スピードを測定する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.9: 悪い測定基準、悪い結果

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 6.10: 責任の所在が違う場所

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 7: 明示的な準備作業（RMF 1）

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.1: Synapse Framework™ との統合

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.2: RMF 1 の構造

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.3: 行動：協働のためのキャパシティを確保する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.4: 成果物：準備作業項目

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.5: アクティビティ：コラボレーション・ミーティング

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.6: 行動：共通理解が得られるまでコラボレーションを継続する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.7: 振る舞い：常にを確認する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 7.8: RMF 1 がをどのように変えるか

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 8: RMF 1 の効果

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 8.1: RMF 1 導入前の状況

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 8.2: 導入前：理解に費やす時間

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 8.3: その後：理解に費やす時間

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 8.4: RMF 1 導入後の生活

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 8.5: 基礎的

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 9: RMF 1 を実践に移す



## 9.1: 教育



## 9.2: チームタイプ別の最低要件



## 9.3: 合意



## 9.4: 準備



## 9.5: パイロット

## 9.6: 展開



## 9.7: フォローアップ



## 9.8: 成功の認定



## 9.9: 継続的な警戒



## 9.10: 「どのように」については？



## 9.11: 実践の時です！

# パート III: 作業完了の承認プロセス

# 章 10: 次なる要件



## 10.1: 解釈の余地



## 10.2: 解釈の余地を狭める



## 10.3: 第三の選択肢：「解釈の余地」なし



## 10.4: 完了への潜在的な影響



## 10.5: 実行への潜在的な影響

## 10.6: 提案する代替案：解釈の余地を残さない



## 10.7: メリット



## 10.8: 分析麻痺への懸念について



## 10.9: 次なるニーズ：カスタマイズされた完了の定義

# 章 11: 一般的な人々の行動



## 11.1: もしそれほど素晴らしいものなら、なぜ皆やらないのか？



## 11.2: 大学からコーチングへのパイプライン



## 11.3: コーチの過剰供給



## 11.4: 完成の定義への一つのアプローチ：しない



## 11.5: 受け入れ基準だけの場合

## 11.6: グローバルな完了の定義

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 11.7: 強制力の欠如

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 11.8: まとめ：実際の完了の定義よりも「DoD」という用語の方が頻出

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 12: 完了の定義を定義する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.1: 一つの作業項目について

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.2: 完了であること

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.3: 正確性

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.4: DoD の構造

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.5: 仕様

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.6: エンジニアリング出口基準

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.7: プロダクト参入基準

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.8: 複数のパーツ、一つのゲート

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.9: 例

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.10: プロセスへのマッピング

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 12.11: まとめ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 13: カスタマイズされた完了の定義（RMF 2）



## 13.1: 原則：すべての作業項目はユニークである



## 13.2: 振る舞い：1 つ以上の完成の定義テンプレートを維持する



## 13.3: アクティビティ：完成の定義テンプレートの定義



## 13.4: 完成の定義テンプレートの維持と改善

## 13.5: 複数の DoD テンプレート



## 13.6: 振る舞い：完了の定義の出発点としてテンプレートを使用する



## 13.7: 振る舞い：カスタマイズされた完了の定義に合意する



## 13.8: アクティビティ：作業項目の完了の定義を定める



## 13.9: ワークフローのさらなる拡張



## 13.10: 振る舞い：実装開始前に DoD を成熟させる

## 13.11: アクティビティ：PBI の完了の定義を成熟させるためのオフライン分析



## 13.12: 成熟化をフローに追加する



## 13.13: 振る舞い：作業項目での完了状態の追跡



## 13.14: 進捗管理の追加



## 13.15: 振る舞い：完了状態による作業のゲート管理



## 13.16: アクティビティ：完了状態の判断に DoD を使用する

## 13.17: ゲーティングの位置づけ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 13.18: まとめ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 14: RMF 1・2 との生活



## 14.1: コスト



## 14.2: タイムライン



## 14.3: 実装チームへの影響



## 14.4: プロダクトオーナーへの影響



## 14.5: リーダーシップへの影響

## 14.6: まとめ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 15: RMF 2 のインストール

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 15.1: ステークホルダーの関与

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 15.2: 詳細レベル

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 15.3: 作業合意

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 15.4: 初期作業

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 15.5: 展開

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 15.6: まとめ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# パート IV: ゲーティング実装

# 章 16: 最終要件

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.1: それはずっとそこにあった

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.2: タイミングの重要性

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.3: 視点の転換

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.4: リスクとコスト

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.5: 準備が整うまで待つことの価値

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.6: 付加的な利点

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.7: 問題提起

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.8: 必要性

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 16.9: まとめ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 17: 準備完了の定義の背景

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 17.1: リーンの作業許可

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 17.2: かんばんのカラム入口基準

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 17.3: スクラムと他のアジャイルプロセスの外伝

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 17.4: サーフィン > コーディング

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 17.5: 部分的な解決策

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 17.6: 準備を始める準備ができている

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 18: 準備完了の定義を定義する



## 18.1: 目的



## 18.2: カスタマイズ



## 18.3: 構造



## 18.4: 例



## 18.5: 合意

## 18.6: ゲーティング

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 18.7: まとめ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 19: カスタマイズされた準備完了の定義（RMF 3）



## 19.1: プロセスにおけるもう一つのゲート



## 19.2: 準備完了の定義の構造



## 19.3: プロダクト出口基準



## 19.4: エンジニアリング入口基準



## 19.5: 重複しても問題なし

## 19.6: 振る舞い：1 つ以上の DoR テンプレートを維持する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.7: なぜ準備完了の定義テンプレートが必要なのか？

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.8: アクティビティ：DoR テンプレートの定義

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.9: 時間とともに DoR テンプレートを維持する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.10: テンプレートは単なる出発点

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.11: 振る舞い：個別の準備完了の定義に合意する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.12: アクティビティ：作業項目の準備完了の定義を定める

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.13: 振る舞い：実装開始前にアイテムを準備完了にする

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.14: チームがコントロールできない条件

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.15: 振る舞い：作業項目における準備状況の追跡

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.16: 振る舞い：準備状況によって作業をゲート制御する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.17: アクティビティ：DoR を使用して準備状況を判断する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 19.18: まとめ

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# パート V: 統合

# 章 20: ほとんどの締切は重要ではない



## 20.1: 本当の締切



## 20.2: 恣意的な締切は重要ではない



## 20.3: 航空会社



## 20.4: 技術的負債の起源



## 20.5: マーケティングと営業の期限

## 20.6: プロジェクトの期限

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 20.7: 別の方法があります

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 20.8: 恣意的な期限は不要

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 20.9: 恣意的な期限は廃止すべき

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 20.10: 実際の期限はまだ要因として存在

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 20.11: *Pis Aller*[1]

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

[1]フランス語に由来する最後の手段を指す言葉。

## 20.12: 結論

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 21: 能力 1：要件成熟化フロー



## 21.1: 下から上へ



## 21.2: 原則：必要な作業すべての透明性



## 21.3: 行動：責任は作業と共に移動する

## 21.4: 振る舞い：要件の状態を可視的に追跡する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 21.5: 振る舞い：準備と実装に関連するすべての作業を明らかにする

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 21.6: アクティビティ：準備作業

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 21.7: 振る舞い：作業項目を完了するために必要なすべての作業を明らかにする

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 21.8: 振る舞い：期限よりも準備を重視する

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 21.9: 結論

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 22: スクラムにおける RMFを用いた作業と情報の流れ



## 22.1: 序文的注記



## 22.2: 初期要件の把握と準備



## 22.3: 準備作業の開始



## 22.4: 準備作業の計画と実行



## 22.5: 準備結果のレビュー

## 22.6: 実装の計画と完了

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 23: RMF の影響

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 23.1: 要件のライフサイクル

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 23.2: 情報と作業のフローの例

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 23.3: 導入前

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 23.4: 導入後

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 23.5: メリット

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 24: RMF への移行



## 24.1: 導入パターン



## 24.2: Scrum ワークフローへの導入



## 24.3: その他のフレームワーク



## 24.4: 主な反発：準備作業項目



## 24.5: 大きな転換：マインドセット

## 24.6: 変革のためのアドバイス

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 24.7: 結論

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 章 25: すべてはあなた次第

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 25.1: 振り返り

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 25.2: さあ、あなたの番です

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# パート VI: リソース

# 付録 A：スクラムが問題なのではない



## スクラムとは何か？



## フレームワーク



## スクラムはプロジェクトと作業の管理に対応する



## スクラムをプロダクトマネジメントフレームワークとして扱うことの愚かさ



### 要件を成熟させるための規定された仕組みの欠如

### 要件開発における技術的専門知識への投資の欠如



### アンチパターン



### 必要な拡張



## スクラムへの追加

# 付録 B: Synapse フレームワーク ™

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## Synapse の対象範囲

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 3 つのマスタリー

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## Synapse の導入方法

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 2 つのフレームワークの融合

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## Synapse フレームワークの構造

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 組織的熟達

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 組織的能力

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 組織的習慣

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# Synapse フレームワークの構造

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## プラクシスの構造

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## 順序の重要性

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

## この本における Synapse の影響

このコンテンツはサンプル版ではご覧いただけません。書籍はhttps://leanpub.com/ready-jaの Leanpub でご購入いただけます。

# 付録 C：RMF 1 に対する一般的な異議と障害



## 一般的な異議



## 一般的な障害

# 付録 D：DoD 初期基準リスト



## エンジニアリング完了基準初期リスト



## 製品参入基準スターターリスト

# 付録 E：DoR スターター基準リスト



## プロダクト出口基準



## エンジニアリング着手基準



## スプリント参入基準

# 索引

20 世紀中頃のプログラミング原則, 7

AI, 20
API, 22

banks, 9
Barreras, Tom, vii

C#/.NET ソリューション, 3
Central Oregon, vii
consultants, 7
culture, 9

dependency services, 9
development environment, 9
domain, 9

Engineering Entrance Criteria, 60
engineering practices, 9
engineering teams, 8, 9

finance, 9

knowledgebase, 8

leadership, 9
Level of Detail, 51
loading up, 8

Mature the DoD, 46

OutSystems, 3

PKB 駆動開発, iv
Producore, iii
Product Exit Criteria, 60
Product Owner, 9

Ready, i, ii
Requirements Maturation Flow, 70
requirements-authoring technique, 9
RMF 1, 30, 81
RMF 2, 51

Scrum, 70
Scrum ワークフローへの導入, 73
Stakeholder Involvement, 51
system, 9

Terminator-style revolt, 20
test-development, vii
The Bank, 12

workflow, 46
Working Agreement, 51

「完了」の定義, 19
かんばん, i
その他のフレームワーク, 73
ほとんどの締切は重要ではない, 65
アジャイル, 24
アジャイルトランスフォーメーション, iv
アップストリームサービス, 11
アメリカの国家金融インフラ, 2
イテレーション, 21
ウォーターフォール, 16
エンジニアリングプラクティス, 15
エンジニアリング幹部, 4
カスタマイズされた完了の定義, 45
カスタマイズされた準備完了の定義, iii, 60
ゲーティング実装, 53
コードベース, 25
コード品質, 24
ゴミを入れればゴミが出る, 15
ゴールドプレーティング, 20
サーバーリソース, 23

システムダウンタイム, 19
スクラム, 5, 16, 21
ステークホルダー, 17
スプリント, 10, 14, 18, 21, 23
スプリント中の変更, 6
セキュリティの脆弱性, 19
ソフトウェア開発, 14, 18, 21
ソフトウェア開発技法, 7
ソフトウェア開発者, 19
チームスキル, i
テスター, 12
デザインパターン, iv
データの損失または破損, 19
バックログアイテム, 24
パズル組み立て, 21
ビヘイビア駆動開発, iv, 7
ピースを数える, 21
フィードバックループ, 13
フロー, 13
プロジェクトの期限, 66
プロジェクトマネジメント協会, 2
プロセスのゲート, 60
プロダクト、役割, 4
プロダクトとエンジニアリング, 14
プロダクトオーナー, 5, 19
プロダクトバックログアイテム, 7, 10, 12, 14, 21
プロダクトバックログアイテムの完了の定義, 47
プロダクトマネージャー, 5
プロダクト組織, 4
マーケティングと営業, 65
リスク, 18
リソース, 76
リーダーシップチーム, 6
レガシーシステム, 6
ローコードソリューション, 3
ローン返済ポータル, 3
ワークフロー, 34

不具合, 24
主な反発, 73
予定より遅れる, 24
人材定着, 25
代替可能な人材, 23
何かが足りない, 1
作業項目, 23
作業項目の完了, 18
信頼, 24

共通理解, 34
初期作業, 51
前提, 24
勝利目前での敗北, 18
反意語同音異義語, 15
大きな転換点, 12
大規模バッチ開発, 16
完了の判断方法, 22
完了の定義, 46
完了状態の追跡, 47
完成の定義, 22
実務担当者, 4
実際の要件, 19
専門家, 6
導入パターン, 73
展開, 51
市場シェア, 19

成功の認定, 37
手戻り, 17
技術的負債, 65
振る舞い：準備と実装に関連するすべての作業を明らかにする, 69
振る舞い：要件の状態を可視的に追跡する, 69
既存のコード, 11
明示的な準備作業, 33
時間の無駄遣い, 20
期限, 21
準備作業項目, 33
**準備完了**, 12
準備完了の定義の構造, 60
準備態勢のための場づくり, 29

無駄, 24
確認された共通理解, 16

累積コスト, 24

自然言語, 15
要件定義, 15
要件成熟化フロー, i, iii, iv, 26, 68
要件成熟化フローへの移行, 73
規制違反, 20
説明責任とトレーサビリティ, 14
追加作業, 19

連邦信用機関, 2
開発チーム, 19
間違った要件, 16
領域専門家, 10