

投稿日 2024/03/29

サービス開発の進め方

本書では、我々が実践してきた新規事業開発におけるサービス開発の進め方を記載します。

概要

プロダクトとしてシステムを開発する場合は、何を作るのかを決めた上で、どう作っていくのかを具体的に設計し、開発を進めます。何を作るのかの言語化については、[プロダクトの言語化](#)をご参照ください。

ここでは、サービス開発の具体的な進め方を解説していきます。

品質への向き合い方

新規事業での開発を進めていくにあたり、どのように向き合っていた方がよいか定めておくべきテーマとして「品質」があります。

変化の多い新規事業開発で、画一的に品質に対する考え方を規定することは難しく、事業ごとに異なる点は出てきます。

ですが、基本となる考えとして以下のようなことを整理しておくといでしょう。

- リリースの方針を定義する
 - たとえば1度にすべて作りきろうとせず、段階的にリリースする（ α 版、 β 版）、など
- リリースごとに主目的を明確にし、そのうえで各リリースに対する品質目標と品質担保戦略を定める
- 可能であれば、ステークホルダーやユーザーからフィードバックを得る機会を設ける
- どのような基準、観点で品質を評価するか整理しておき、各リリースでどのように取り組むかを定める
- コード品質、ユーザビリティ、パフォーマンスなど

品質に対するこのような考え方を事前に整理しておき、方針として定めておきましょう。

これを行わないまま開発を進めてしまうと、「品質」の定義、捉え方が人によってまちまちとなり、エンジニア間やステークホルダーとのギャップが生じる原因となるでしょう。

たとえば、我々が品質戦略を十分考えずに開発を進めた結果として、以下のような問題が発生しました。

- 対象のリリースで、事業で求められる品質観点を担保できていなかった
- 今回のリリースでは、求められていない品質を追求して時間を使ってしまった
- リリース速度と品質を天秤にかける事態となった時に、今回は妥協してもよい品質観点を判断できなかった
- エンジニアの間で重視する品質観点が合わず、レビューなどで大きく工数を使ってしまった

このような齟齬をなくしてリリースに向けた品質を担保できるよう、品質戦略を立て関係者間で合意しましょう。

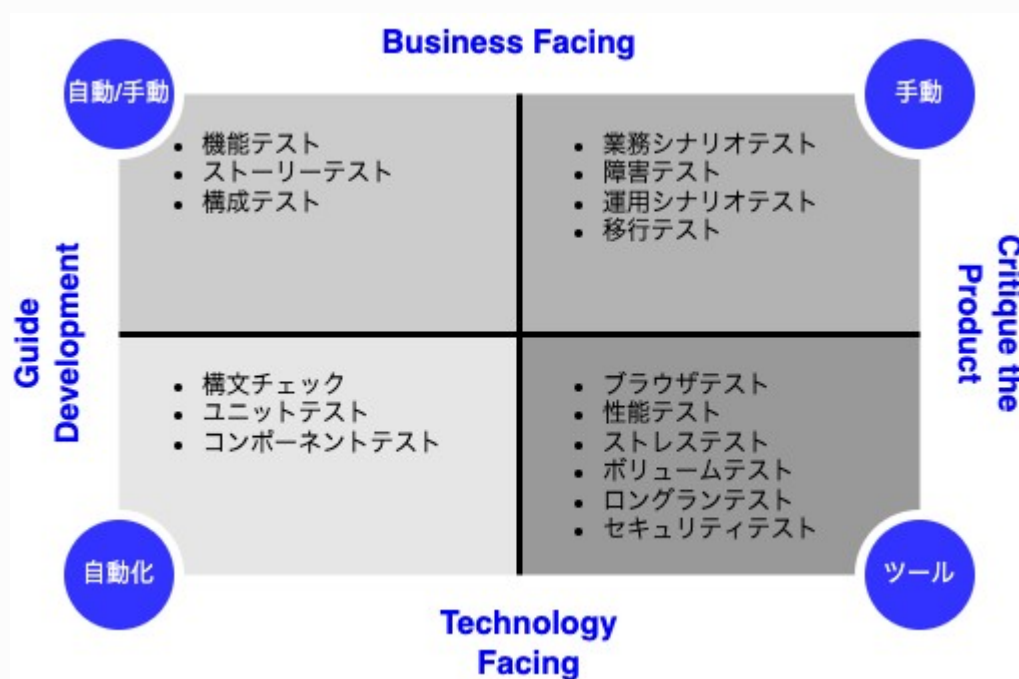
全体テスト計画

限られたリソースの中で効果的・効率的にプロダクトの品質を担保するためには、テスト全体としての計画を立案し、「各テストで確認すべき品質の明確化」と「各テストでの品質の積み上げ」を行わなければなりません。

そのためにはまず、毎スプリントでどのテスト観点を担保するのか、テストスプリントでどのテスト観点を担保するのかを明確にすべきです。これは、各スプリントで担保するテスト観点を多くするほど各スプリントのインクリメントの品質は担保される一方、

テストに要する時間が増えていく関係にあるため、適切に判断しないとスピーディな開発ができなくなるためです。

アジャイルなプロジェクトにおけるテストは、「開発支援」「プロダクト評価」「ビジネス観点」「技術観点」からなる「アジャイルテストの4象限」¹で表現できます。下記の図のなかで示している個々のテスト種別については、[テスト種別カタログ](#)を参照ください。



アジャイルテストの4象限

基本的な方針としては、毎スプリントでは「開発支援」に該当する左側の象限のテストを実施しましょう。右側の象限にある「プロダクト評価」についてはプロダクトの全体像をある程度作り上げた段階でないとテストを実施できないため、主としてテストスプリントにて実施しましょう。ただし、リリース直前まで大きなリスクを内在したままになることを避けられるように、できるだけ早い段階で露払いを行うことをお勧めします。

Done・Undoneの定義

毎スプリントで担保するテスト観点を含め、個々のスプリントで何を行い、何を行わないのかを表現したルールが「Doneの定義とUndoneの定義」です。スプリントを開始する前にこれらを作成し、スクラムチームで合意しましょう。

定義の目的や意義については[Doneの定義](#) | [アジャイル・スクラム](#) | [Fintan](#)を参照ください。

開発プロセスの選択

MVPでは成立したはずの仮説が、プロダクトをリリースしてみたら間違っているということも新規事業開発では起こり得ます。こういった場合、開発に多大な時間とお金をかけてプロダクトをリリースしても、事業継続が困難になります。

このため、短いスパンでプロダクトをリリースし、ユーザーに使ってもらい、手応えを掴みながら改善を繰り返さなければなりません。

これはまさにアジャイルな手法が必要とされるケースであり、我々は、その実践手法としてスクラムを選択しています。

スクラムの概念や考え方についての詳細は[スクラム概論](#)をご覧ください。

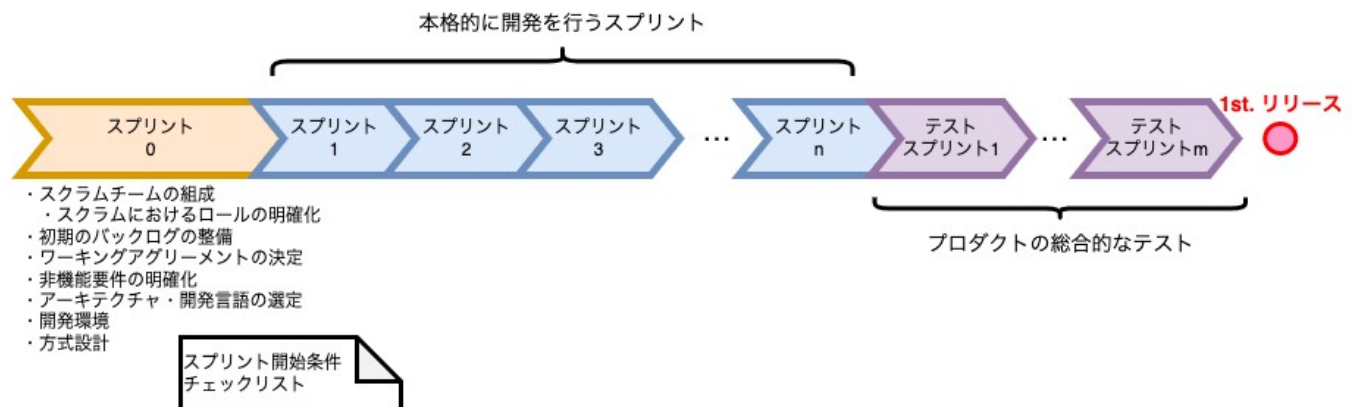
開発を円滑に進めようと思うと、まずはスクラムの基本となる考え方を関係者全員が理解し、開発をチームで自律的に推進できるようになる必要があります。

計画

新規事業開発における開発の全体像は下図で示す形になります。

この全体像を踏まえて開発工数を見積もり、意思決定者やマネジメント層に伝えましょう。見積の方法については「[プロダクト機能の見積](#)」を参照ください。

1st. リリースまでのスプリント

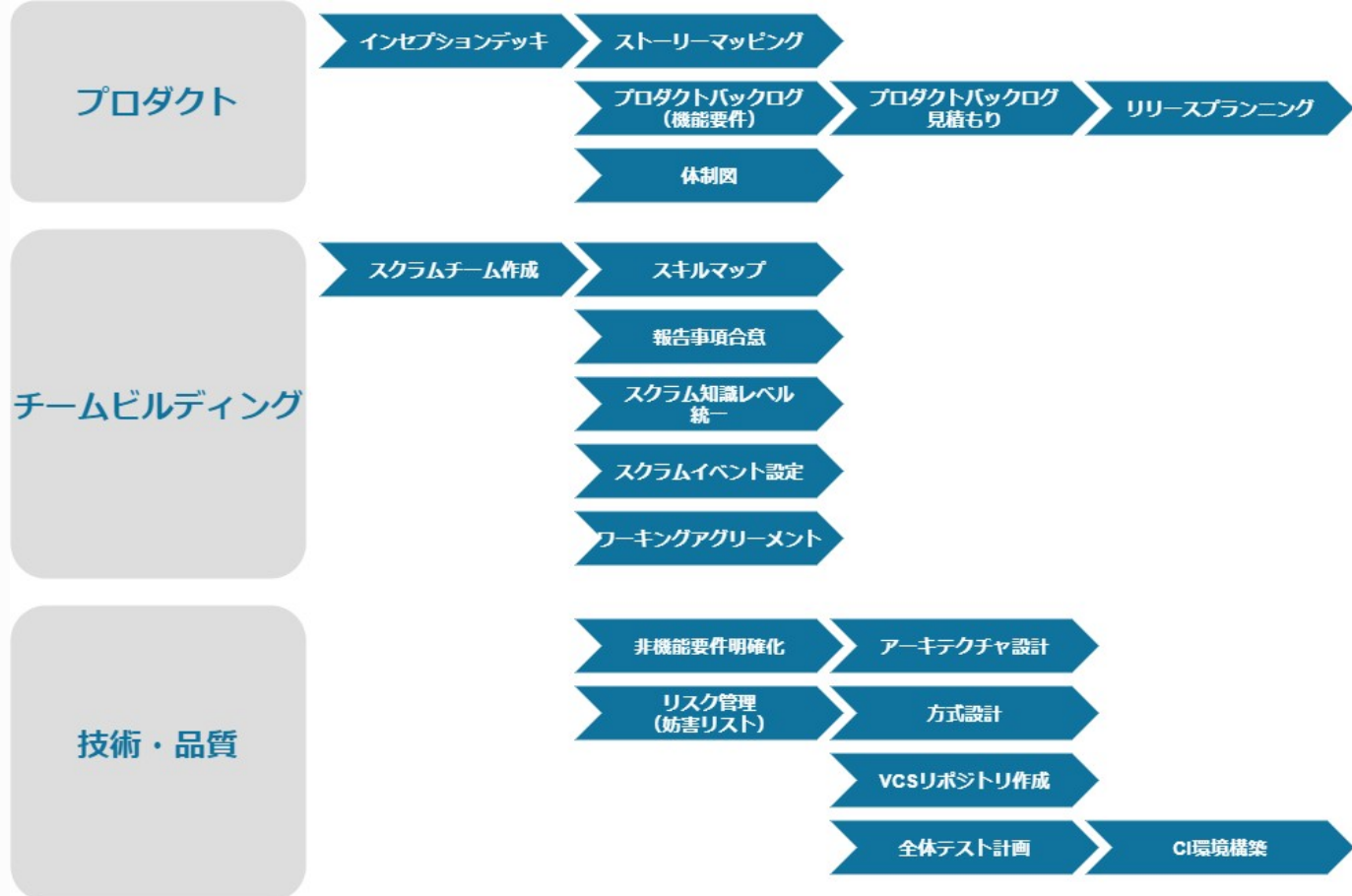


1st. リリースまでの全体像

スプリント0

スプリント0は、本格的な開発を始める前の準備期間です。

準備すべき内容については以下の図を参照してください。



スプリント0でやるべきこと

各項目の具体的な進め方については[Scrumプロジェクト開始のベストプラクティス #RSGT2018](#)を参考にしてください。

全体テスト計画、Done・Undoneといった、特に我々が重要と考えているアクティビティの定義については後述します。

テストスプリント

リリース時のプロダクトが「出荷可能」な品質になっていることを確認する期間です。

注意していただきたいのは、テストスプリントがあるからといって通常のスプリントで品質担保を疎かにすべきというわけではないことです。スプリント単位でおよそ問題ないと言える程度の品質を担保しなければ、テストスプリントで大量の不具合が発覚することになります。

進捗管理

スクラムで進める開発であっても、「いつリリースできるのか」という問いには答える必要があります。

そのためには、ある程度の長期にわたる計画を立て、その進捗状況をトラックしなければなりません。

このような計画の立案を「リリースプランニング」と呼びます。

リリースプランニングを行う上ではリリースの制約を明らかにする必要があります。主な制約となるのはスコープ、期日でしょう。どちらを制約とするかによって、開発の進め方が変わります。

リリースの制約	スコープ	期日	予算	説明
スコープを固定	固定	可変	可変	時間を使い果たしたときに、事前に定められたスコープが完成していない場合、期日を延期する。
期日を固定	可変	固定	固定	優先度の高い機能から開発する。期日が来た場合、完成している機能をリリースする。

この判断は、継続的に行う必要があります。後述する進捗の可視化で進捗上の問題が検知された場合に、見直しを行います。

固定スコープのリリースプランニングと進捗管理

スコープが固定の場合のリリースプランニングは以下の手順になります。

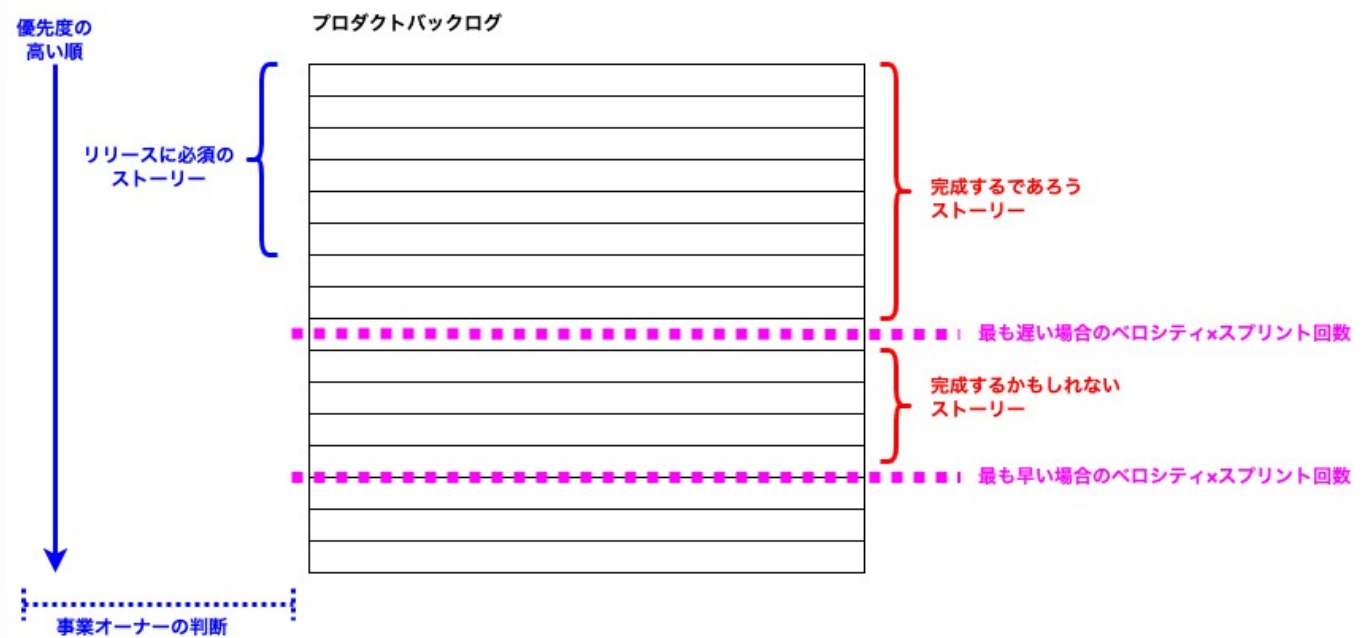
1. スコープに含まれるプロダクトバックログアイテムを詳細化・見積を行い、合計サイズを算出する
2. チームのベロシティの範囲を決める
3. 合計サイズを最も遅い場合のベロシティで割り、端数を切り上げる。これがリリースに最低限必要なスプリント数となる
4. 合計サイズを最も早い場合のベロシティで割り、端数を切り上げる。これがリリースに必要なかもしれない最大のスプリント数となる

このリリースプランニングにより、「対象スコープの開発を完了させるためには○回から○回のスプリントが必要になる」という答えが得られます。不確かさが残るためにリリース時期は「範囲」になりますが、時間が経つにつれて徐々に正確になっていきます。

固定期日のリリースプランニングと進捗管理

期日が固定の場合のリリースプランニングは以下の手順になります。

1. リリースまでのスプリント回数を計算する
2. プロダクトバックログを適切な詳細度まで分割し、優先順位を決める
3. チームのベロシティの範囲を決める
4. 最も遅い場合のベロシティと1.のスプリント回数を乗じ、そのポイント分のプロダクトバックアイテムを選択する。これが「完成するであろう」ラインになる
5. 最も早い場合のベロシティと1.のスプリント回数を乗じ、そのポイント分のプロダクトバックログアイテムを選択する。これが「完成するかもしれない」ラインになる



固定期日の場合のプロダクトバックログ

これにより、期日までにどこまでできあがるのかの答えがある程度わかります。ここでも不確実さは排除できませんが、不確実さの程度は次第に小さくなっていくはずです。

設計

アーキテクチャ設計は、非機能要件と業務要件に左右され、事業ごとの設計が必要になります。

設計ドキュメントについては「[プロダクトの言語化](#)」をご参照いただくこととし、ここでは設計上で押さえておかねばならない重要なポイントのみに絞って記載します。

最終的なソリューション像を描いた上で設計を実施する

アーキテクチャ設計で重要なのは、ビジネスにどのような機能要件・非機能要件が求められるのかを正確に把握し、それをアーキテクチャに落とし込むことです。

この「アーキテクチャに落とし込むべき要件」は、1st. リリースで必要になる要件ではなく、サービスの将来像を見越した要件であることに注意しましょう。

これはアーキテクチャに落とし込むときの判断は時に「不可逆」な判断となり、あとで修正が効かない、あるいは修正するのに多大なコストがかかるためです。

保守性を強く意識する

改善を積み重ね素早く成長させていく新規事業においては、保守性を強く意識しなければなりません。

保守性を犠牲にして短期的なスピードを得たとしても、中長期的には逆効果になります²。以下の資料も併せてご参照ください。



アジリティを支える 品質特性

Jul 30, 2021 @ デブサミ2021夏



和田 卓人 (@t_wada)



環境を分離する

セキュリティの担保、影響範囲の限定、リソースの効率的な利用などの観点から、本番・開発といった環境は明確に分離しましょう。

AWS Well-Architected Frameworkにも、「AWS では、アカウントが強固な境界となります。

例えば、開発およびテストのワークロードと本番ワークロードを切り離すために、アカウントレベルの分離を強く推奨しています」³ という記載があり、環境分離が推奨されています。

詳細は[AWS アカウントの管理と分離 – セキュリティの柱](#)をご覧ください。

開発プラクティス

バージョン管理

VCSを利用したアプリケーションコード等のバージョン管理は必須です。我々はGitを利用しています。

管理対象には、コードだけでなく構成管理用のファイル（たとえば、Javaであればpom.xml、Node.jsであればpackage.jsonなど）も含みます。利用ライブラリを管理することで、後述する脆弱性管理やバージョン管理も容易になります。

ブランチ戦略

我々が利用するのは[GitLab Flow](#)です。

このワークフローは、ユーザー機能駆動開発(Feature Driven Development)とフィーチャーブランチ(Feature Branch)といった開発プロセスを含んでいます。

プルリクエスト駆動開発

新規事業開発におけるプロダクト開発では、アプリケーションの変更頻度が高くなります。

チーム内のピアレビューにてその品質を担保しましょう。ピアレビューは、変更の品質を向上させるだけでなく、相互訓練、ピアラーニング、スキル向上の効果を持っています⁴。

Infrastructure as Code (IaC)

インフラ開発でも、バージョン管理、ブランチ戦略、プルリクエスト駆動開発を実践しています。そのためにはIaCの利用が必須となります。

手作業でインフラを構築すると、最初の段階では一貫性を保ちながら構築・設定できたとしても、時間が経つにつれて一貫性が失われ、管理も難しくなります。これは、絶えず変化が必要な新規事業開発にとって大きな足枷になります。

継続的インテグレーション(CI) / 継続的デリバリー(CD)

CI/CD は、「アプリケーション開発の各ステージに自動化を導入し、顧客にアプリケーションを頻繁に提供できるようにする手法」⁵です。CI/CDを通じて、アプリケーションのデプロイまでのライフサイクルに、継続的な自動化と継続的な監視を導入できます。

静的解析

静的解析では、ソースコードや設定ファイルを解析し、特定のコーディングパターンに違反している箇所をチェックします。

これにより開発の極めて早い段階でエラーを検出できるとともに、優れたコーディングスタイルを推進でき、可読性や保守性も向上します。

また、機械的に指摘できるエラーを摘み取ることもできるため、ピアレビューではより高度な問題にも集中できるようになります。たとえばJavaには、[SpotBugs](#)や[Checkstyle](#)といった静的解析ツールがあります。

静的解析も含めた具体的な実装例として、[example-chat](#)を公開しています。

ユニットテスト

アプリケーションのコードが期待通りに振舞うことを検証する基本となるのがユニットテストです。継続的インテグレーションがもたらす核でもある「素早いフィードバック」は、ユニットテストのカバレッジが十分ないと可能になりません。素早く実行でき、カバレッジの高いユニットテストを記述していきましょう。

具体的なサンプルについては、[GitLab CI/CDでの静的解析を含むステップの定義](#)をご参照ください。

脆弱性管理

プロダクトのセキュリティを担保するためには、脆弱性の早期発見と修正が必要です。脆弱性の早期発見には、静的解析やユニットテストの他に、ライブラリの脆弱性チェックや脆弱性スキャナーの利用が有効です。

脆弱性チェックとしては、[npm-audit](#)や[OWASP Dependency-Check](#)を利用することで開発上大きな負担なく脆弱性チェックが可能です。我々は上述のCIにこれらのツールを組み込んでいます。

また、ライブラリ単体ではなく、プロダクト全体としての脆弱性チェックも重要です。我々はIAST (Interactive Application Security Testing)と呼ばれるタイプの脆弱性スキャナーを利用し、基本的な脆弱性を開発中に発見できるようにしています。

まとめ

新規事業開発におけるサービス開発の進め方について記載しました。

これらの手法を実践することで、開発の効率化や品質の担保がしやすくなります。また、これらの手法は、我々が実際に実践してきたものであり、その効果を実感しています。

1. [Using Models to Help Plan Tests in Agile Projects\(2022年3月13日閲覧\)](#)↗
2. [「品質の高いソフトウェアはそのコストに見合うのか？」\(2022年3月11日閲覧\)](#)↗
3. [AWS アカウントの管理と分離 – セキュリティの柱 \(amazon.com\)](#) より引用↗
4. ジーン・キム,ジェズ・ハンブル,パトリック・ポア,ジョン・ウィリス、[『The DevOps ハンドブック 理論・原則・実践のすべて』](#)、日経BP、2017年。↗
5. [CI/CD とは-継続的インテグレーション/継続的デリバリー|Red Hat](#) より引用↗

/* Recommend */

「新規事業開発」 のおすすめ記事はこちら

この記事に関連する記事もお読みください。

新規事業開発


プロダクトの言語化

2024/03/29  3


新規事業開発

仮説検証とMVPへの向き合い方

2024/03/29  2

新規事業開発

新規事業開発のステージ・ゲート
プロセスとその評価基準

2024/03/29  21

最近投稿された記事も用意しました。

新規事業開発

サービス開発の進め方

2024/03/29  4

新規事業開発

プロダクト機能の見積


2024/03/29  2

新規事業開発


プロダクトの言語化

2024/03/29  3

「新規事業開発」 で最も読まれている記事を以下にまとめています。

新規事業開発

新規事業開発のステージ・ゲート
プロセスとその評価基準

2024/03/29  21

新規事業開発

新規事業スタートガイド

2024/03/29  48

新規事業開発

仮説検証とMVPへの向き合い方

2024/03/29  2