ソフトウェア品質シンポジウム 2023
B3-1 2023-09-08

QAチームを作る前に実施できるプロダクトバックログアイテムの見直しによる品質向上

株式会社カケハシ

○ 笹尾 納勇仁

n.sasao@kakehashi.life

KAKEHASHI

Mission

日本の医療体験を、しなやかに。

KAKEHASHI

高潔

価値貢献

カタチにする

無知の知

変幻自在

情報対称性

KAKEHASHI

Musubi
AI在庫管理

KAKEHASHI

発表概要

- Musubi AI在庫管理プロダクト開発チームでの取り組み
- 魅力的なプロダクトを継続的にリリースするため
- プロダクトバックログアイテムの記載内容の見直し
- 実際に取り組んだ内容や工夫を具体的に紹介
- ユーザー影響が大きなバグの低減などの成果を具体的に紹介

2023年2月に入社してから
取り組んだことと成果を具体的に紹介します

KAKEHASHI

背景

KAKEHASHI

スタートアップとしての背景

- プロダクトが安定して市場に受け入れられるまで
  - 人員を増やすことが難しい
  - 継続的な機能リリースを求められる
- プロダクトが成長しユーザーが増えると品質問題も顕在化する
- 開発チームの増員ができたとしても
  - 優先すべきは機能リリース
  - QAチーム立ち上げまで手が回らない

KAKEHASHI

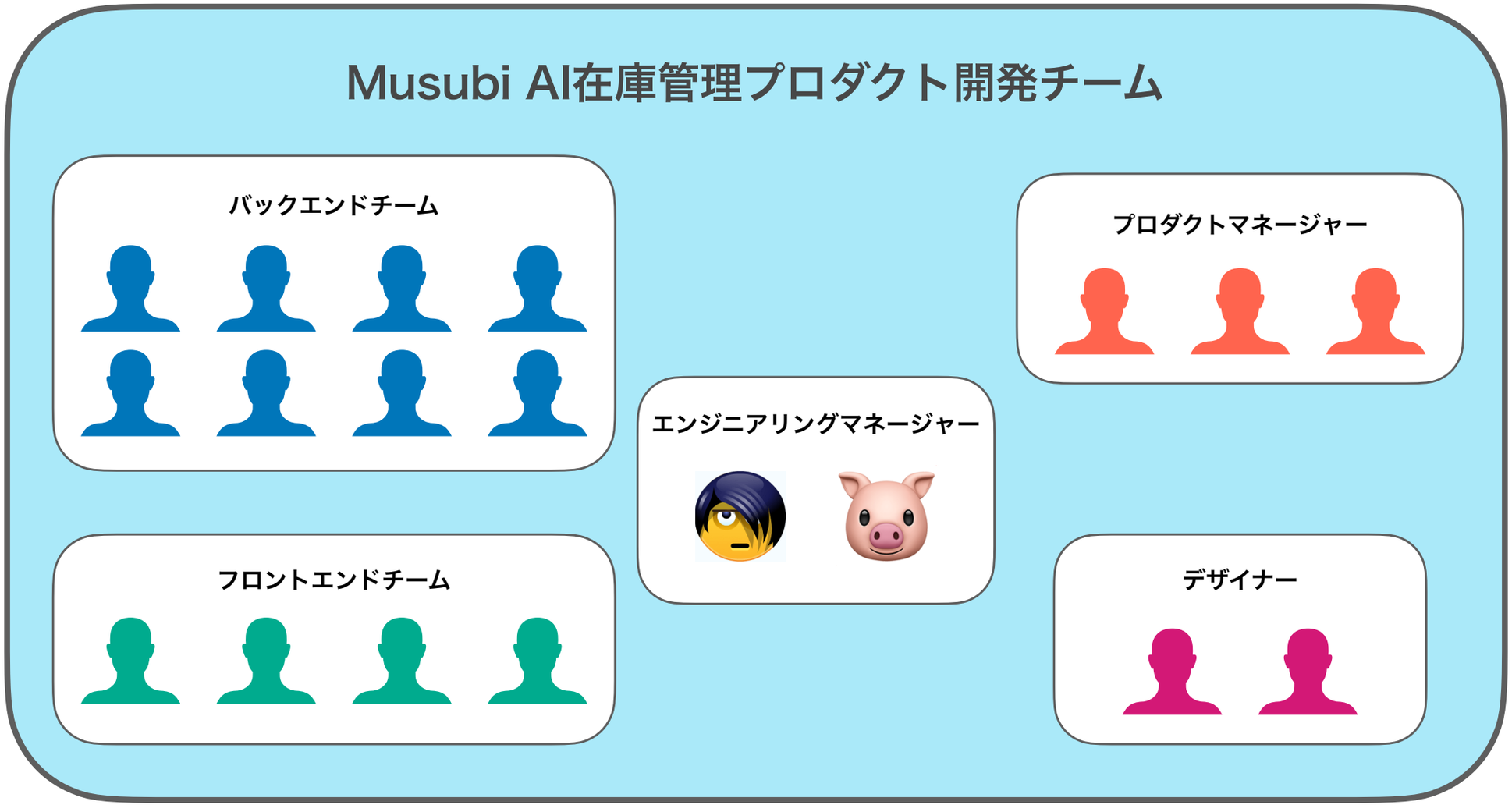
プロダクトのフェーズとしての背景

- BtoB SaaSの 0-1フェーズから1-10フェーズへ
  - SMBへの導入は順調に進み始める
  - Enterprise領域にも狙いを定めるタイミング
- Enterprise領域は既存運用を踏襲する必要あり
  - コア機能であるAI部分とは直接関係ない機能が不足
  - 競合となる在庫管理システム同等の機能が必要

KAKEHASHI

チーム構成としての背景

チームは拡大とともに
スキルやロールによる分業化が進み始めている



KAKEHASHI

解決すべき課題

KAKEHASHI

障害発生とその内容

- 2/1から4/30までに発生した障害 5件 について
  - 3件は要求の合意漏れ
    - うちお客様の業務に回避不可能な影響が発生した高いレベルの障害が2件

ユーザーストーリーの完了条件記載による合意が可能だが
完了条件に合意すべき要求の記載なし

期間	発生日	内容	障害レベル	原因
2023/2/1 〜 2023/4/30	2023/2/10	おすすめリストが正しく表示されない	**高い**	要求の合意漏れ
	2023/2/26	エラーにより後続の発注おすすめの計算処理が実行されない	**高い**	AWS内のエラー
	2023/4/4	合計数量を算出するコンポーネント内のエラーでアプリが落ちる	低い	要求の合意漏れ
	2023/4/18	入庫原価が空欄で登録さてしまう	**高い**	要求の合意漏れ
	2023/4/21	在庫調整を利用した際に正しくデータが送信されていない	**高い**	期間外の実装

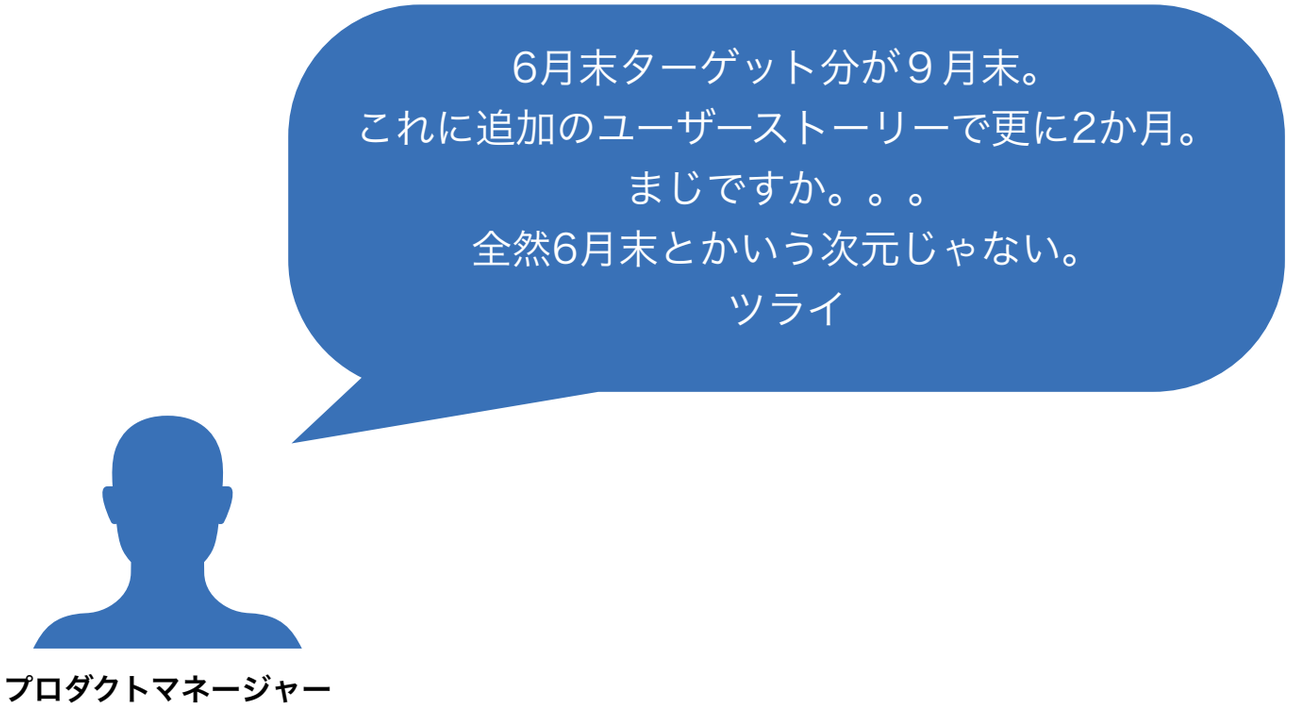
※ 期間外の実装は2023年1月以前の実装起因によるもの

KAKEHASHI

スケジュール遅延の発覚

ユーザーストーリーの内容が大きいことで
認識齟齬と見積り漏れに繋がりスケジュールとのズレが発生

4月後半に仕掛かり中のユーザーストーリーについて
大幅な予定変更を強いられる



KAKEHASHI

実施検討

KAKEHASHI

基本方針

- 前提条件
  - 開発人員増やさず専任の担当者がいなくても実施可能
  - 機能開発は継続的に実施すること
- 実現したいこと
  - 障害の低減（特にお客様の業務に回避不可能な影響のあるもの）
  - スケジュール遅延の低減
  - プロダクト品質に対するチームの認識を向上

KAKEHASHI

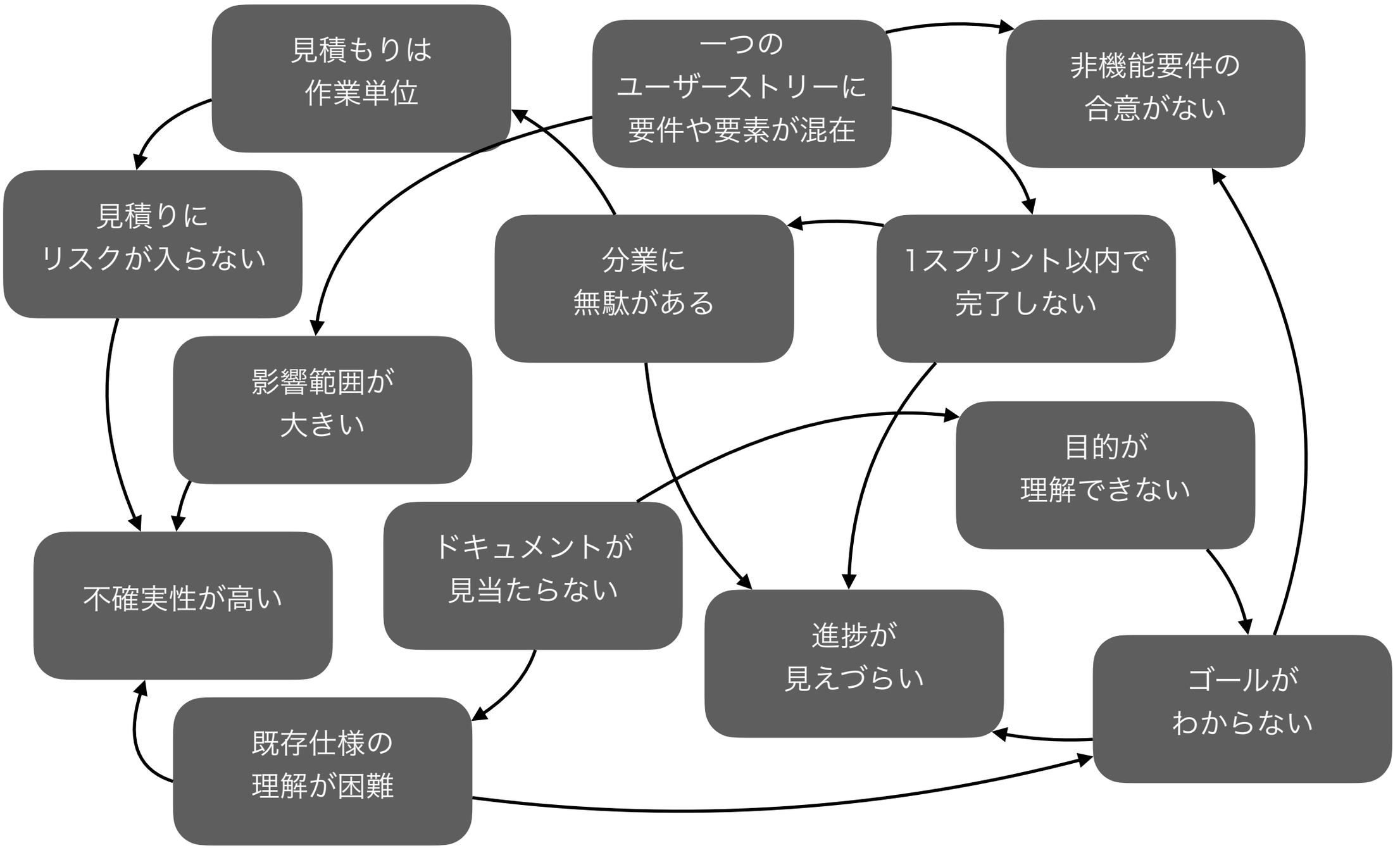
自分が2月に入社して感じたこと

- JIRAで管理されたユーザーストーリーを見ても価値や目的がわからない
    - 要求の経緯が記載された資料が別にあるが、最終的な要件は記載されていない
    - Githubのissue、Figmaのデザイン、複雑な既存仕様を理解しないと要件がわからない
- JIRAで管理されたユーザーストーリーの完了が何が実現できたら完了なのかわからない
    - ユーザーストーリーに関連するGithubのissueが完了するとユーザーストーリの完了になる
    - Githubのissueだけ読んでも意味がわからない
    - 非機能要件など要求の合意すべき記載がどこにも見当たらない
- 既存仕様の理解が難しい
    - ドキュメントが少なくまとまっていない上、最新がどれかわからない
    - 仕様の理解に役立つテストコードが少ない
- JIRAで管理されたユーザーストーリーの見積もり根拠がわからない
    - Githubのissueに記載した作業単位で見積もりしている
    - ユーザーストーリーの見積りにリスクが考慮されていない
- JIRAで管理される一つのユーザーストーリーの進捗がわからない
    - 一つのユーザーストーリーがスプリント内で終わらない
    - 毎日進捗確認をしているGithub projectで管理されたissueのみ
- JIRAで管理される一つのユーザーストーリーの不確実性が高い
    - 変更に対する影響範囲が大きい
- 情報の連携が見えにくい
    - 分業化が進みロールごとに作業を進めている
    - 同じユーザーストーリーなのにロールが違うと違う期間に開発している
    - デザイナーは情報設計してから1.5ヶ月以上先行放置されているユーザーストーリーが複数ある
    - 検討開始から参加したメンバー以外が目的や内容を詳しく理解できない

開発プロセスが見えない
開発内容が理解できない
進捗把握も難しい
認識齟齬が不安

KAKEHASHI

一つ解決しても上手くいきそうにない



好ましくない現状が好ましくない現状を生み出している

KAKEHASHI

実施検討

機能開発を継続しながら実施するには
今やっている事を全員が理解しやすくして
認識齟齬を減らす必要がある

- 障害の低減（特にお客様の業務に回避不可能な影響のあるもの）
  - ユーザーストーリーの完了条件に要求を記載すれば合意漏れによる障害は低減するかも
- スケジュール遅延の低減
  - 1スプリントで確実に完了可能なユーザーストーリーであれば進捗が見えやすいかも
  - ユーザーストーリー自体が小さければ認識ズレが減らせるかも
- プロダクト品質に対するチームの認識を向上
  - 求める品質をユーザーストーリーに明示的に記載することで認識可能

今やっていること、これからやることを管理する
プロダクトバックログの見直しが有効だと考えられる

KAKEHASHI

実施内容

実施対象期間は2023年5月1日から2023年7月31日の3ヶ月
比較対象期間は2023年2月1日から2023年4月30日の3ヶ月

KAKEHASHI

実施概要

- プロダクトバックログアイテムの見直しによる品質向上
  - ユーザーストーリー （実際の開発アイテム）
    - 記載内容の見直し
    - **価値**分割の実施
    - 完了条件の記載
  - エピック （ユーザーストーリーの親となる要望）
    - トレードオフを管理
- 見積もりを通したユーザーストーリーの認識合わせ
  - ロールを跨いだ見積もりとユーザーストーリーの合意

ユーザーストーリーにおける価値とは何か？

KAKEHASHI

品質と価値の繋がり

品質とユーザーストーリーにおける価値の繋がりを見つける

品質が高い ≒ 価値が高い

価値 ≒ 得られる体験や利益

価値が高い ≠ バグがない

価値が高い ≠ 工数が大きい

少ない工数で高い価値を生み出せる状態を目指す

KAKEHASHI

実施準備

プロダクト開発チーム全員が
プロダクトバックログを通して価値を考えれるよう
記載ガイドを用意

記載方針をガイドに記載しいつでも閲覧可能に

KAKEHASHI

ユーザーストーリーのタイトルに価値を明記

新たに入ったメンバーや異なるロールでも
共通認識が持てる状態を目指す

タイトル

『誰が』、『どんな理由で』、『どんな価値を』、『どんな状態で』、享受できるのかを記載する。
また、ユーザーストーリーのユーザーは顧客であるかに関わらず、プロダクトに関わる全ての人が対象になる。

大切なのは、この記載でこのストーリーが完了した時に、どんな状態になっているのか？
次に関わる誰かユーザーセグメントを明確にした上で、どんな体験を得るか？
ユーザーストーリーの完了後の様子がわかるように、自工程完結の意味合いも含めて単体の作業ではなく、必要な物事のボリューム感がわかることを意識する。

得られる価値について

通常考えやすいもの以外については、以下やそれ以外も考慮する。

- 体験できる
- 知ることができる
- 共有することができる
- フィードバックを得ることができる

得られる体験を含めた価値の種類まで記載

KAKEHASHI

タイトルの記載のBefore/After

価値や目的も機能もよくわからない

ほとんど同じタイトルのユーザーストーリーがいくつもあり
「出庫処理と同様」のため出庫処理を知らないと全く意味がわからない

~~HAK-467~~	frontend_HAK-1739-1_在庫調整において出庫処理と同様の形式で入庫処理を実施することができる（法人間融通Ph.3）	39.5		完了
~~HAK-2279~~	backend_HAK-1739-1_在庫調整において出庫処理と同様の形式で入庫処理を実施することができる（法人間融通Ph.3）	54.5		完了
~~HAK-3042~~	backend_HAK-1739-2_在庫調整において出庫処理と同様の形式で入庫処理を実施することができる（法人間融通Ph.3）	42		完了
~~HAK-3041~~	frontend_HAK-1739-2_在庫調整において出庫処理と同様の形式で入庫処理を実施することができる（法人間融通Ph.3）	15		完了

価値や目的が明確で機能も想像しやすい記載へ

HAK-6464	出庫元の店舗経営者は、取引先管理の法人間融通のデフォルト設定に適格請求書発行事業者の登録番号と登録日（登録をうけた日）を登録する(且つ必要に応じて...	-		TO DO
HAK-6465	適格請求書発行のための設定実施者は、設定済みの登録番号を取引先管理の法人間融通のデフォルト設定欄で確認することができる	-		TO DO
HAK-6473	適格請求書発行のための設定実施者は、設定済みの登録日を取引先管理の法人間融通のデフォルト設定欄で確認することができる	-		TO DO
HAK-6470	適格請求書発行のための設定実施者は、取引先管理の法人間融通で登録番号の反映範囲を表示することで、適格請求書発行のための設定箇所を正しく理解できる。	-		TO DO

KAKEHASHI

ユーザーストーリーの価値分割

小さなリリースによって価値を積み重ね
進捗を見えやすくすると同時に
品質要件や品質要素の混在を減らし完了時の状態を明確にする

ユーザーストーリーの分割

ユーザーストーリーはできる限り最小の単位となることが好ましい。
分割の例として以下がある。上から順にとにかくやれるだけやってみる。

- データの境界に沿って分割する
- 操作の境界で分割する
    - 登録、変更、削除は別の価値
    - 1項目の操作だけでも価値になるよ
- 横断的な関心事を分割する
- パフォーマンス制約をストーリーにする
- 優先度に沿ってストーリーを分割する
- タスクを価値のある単位でまとめたり、価値のあるタスクを切り出してストーリーにする
- 品質要素や品質要件の境界で分割する
- ユーザーのセグメントで分割する
- 完了条件の一つを切り出す

注意点としては以下がある

- 関連する変更への誘惑は断ち切ってとにかく小さくすることが大切
- 調査、設計などが大きくなる場合は、発見や知ることに価値を置いてなにがわかればいいのか定義する
- 共通認識を持てるための何かを生み出すのも価値になる

分割基準を細かく記載し価値分割を行いやすくする

KAKEHASHI

価値分割のBefore/After

1スプリント(2週間)で完了しないサイズのため進捗も見えづらい

40ポイント以上のストーリーが
1スプリントのベロシティは15ポイント程度しか消化できない

1スプリント(2週間)で1つから2つ完了可能な大きさに

KAKEHASHI

完了条件の記載

異なるロールでも
理解できる形式で
完了した時の状態を記載し
認識齟齬を減らす

記載内容が思いつかない場合の
ヒントをガイドに用意

- 何が見えたらいいのか？
  - データ
  - コントロール
- どんな情報を許容してどんな情報は許容しないのか？
  - 現在のユーザーの情報を元にした〇〇のパターンなど
- どんな体験があればいいのか？
  - 表示速度
  - 操作感
  - どんなユースケースが満たされるのか？ →大きい場合はエピックに
- どんな動作をすればいいのか？
  - どうしたら保存されるのか？
  - どうしたら更新されるのか？
- データの有無によって何が変わるのか？
  - UIに関わらずUI stuck等を参考に広げるのが良い ※下図参照
- どんな資料があって後から何が分かればいいのか？
  - HMWのような後に残る概念資料が必要か
  - ユーザーに明示する程度の画面の仕様書が必要か
  - アーキテクチャ図が必要か
  - フローが必要か(データフローなど)
  - ピヘイビアを表すテストなど必要ないか？
- どんな非機能要件が担保できればいいのか？
  - どれぐらいの失敗は許容できるのか？
  - どれぐらいで回復すればいいのか？
  - 何が明確になっていればいいのか？
- どんな運用が回ればいいのか？
  - 手作業の運用があるのか？
  - 顧客業務を停止する必要があるのか？
  - 異常発生時にどんな通知が必要か？
- 継続的な品質に関して
  - リグレッションテストとして担保したいことは何か？
  - 今回のリリースに影響しやすい事柄が何か？
  - 設計上避けるべき影響は何か？

KAKEHASHI

完了条件のBefore/After

完了条件は全く記載されていない

説明もないためユーザーストーリーの詳細を見る意味を感じない

完了条件がチェックリストとして記載されており
確認後にクローズしやすくなっている

KAKEHASHI

エピックごとのトレードオフ管理

ユーザーストーリーの親となるエピック定義は
要望としてのトレードオフを明確にすることで
限られたリソースで判断すべき方向性を指し示す

トレードオフの明記

基本的に以下のトレードオフを明記する。
途中でトレードオフを変更する場合があっても構わないが、変更したタイミングから達成できる範囲での変更にとどめること。
要求度の平均が必ず3になるように設定し、できる限りバラバラの数値を設定すること。
全て3になるのはNG

下記の例では、少ない工数を維持するためにスコープを絞り、期日は大体その辺りにリリースできたら嬉しい程度の事がわかる

対象	要求度	低　1	2	3 中	4	5　高
コスト(予算に応じて工数を絞る)	-					○
デリバリー(期日を守る)	-			○		
スコープ (仕様の範囲を守る)	-	○				

KAKEHASHI

トレードオフ管理のBefore/After

ユーザーストーリーが大きく柔軟なスコープ調整が困難
コストはエピックに対する予算などの基準がなく判断材料にならない
計画に問題があるとデリバリーへの影響が避けられない

トレードオフが成立していない

ユーザーストーリーを小さくしてスコープの調整を容易にしつつ
共通のコスト感を養う方針を明確にして進める

KAKEHASHI

ロールを跨いだ見積もりと ユーザーストーリーの合意

見積り時に納得するまでユーザーストーリーを会話し

コスト認識の違いを見つけて認識齟齬を減らす

目的

本見積もりで、スコープに対する価値を明確にして、見通しやスケジュール(短期-中期のロードマップ)を立てやすくすることが第一の目的。

一方で今後のプロダクト開発チームとして、クロスファンクションや異なるロール間での見積もりを行い、それぞれが予実で振り返りコスト感覚を合わせることで
今後の計画を立てやすく合意しやすくすることも目的としている。
小さい見積もりで合意できなければ、大きなサイズも難しいため、まずは小さな見積もりからチャレンジしていく。

また、タスクではなくユーザーストーリーの見積もりを通して、価値やコスト感をすり合わせることで隠れたリスクや、隠れた解決方法が見つかることもあるのでしっかり納得感を得るまで会話する。

手順

1. あくまでユーザーストーリーがあるものだけ積算見積もりします。
    - 概算見積もりは各ユーザーストーリーの説明文に記載
2. 合意可能なユーザーストーリー(2-3人日以内目安で合意しやすい)単位の洗い出しを行って見通しを立てます
3. 外部の資料がなくてもユーザーストーリーを見たら価値や完了条件がなんとなくわかり完了時の状態が想像でき、そのためのリスクなど話あえる状態であること
    a. もしユーザーストーリーを洗い出ししても合意可能な完了条件を見つけられない場合や、そもそもユーザーストーリーが立てれない場合
        i. その不確実性を排除したと確認できる合意を完了条件に明記したユーザーストーリーをたて、予算を立てエイヤーで見積もる
        ii. このユーザーストーリーの価値を分割し、記載し直してこのユーザーストーリーはdoneにし、分割後のストーリーに後半の完了条件をいれる。
4. もし見通しとなるユーザーストーリーが全て出揃って合意の条件となる完了条件の概要まで出揃ったら、各ユーザーストーリーの予算とエイヤー見積を立てる
    a. 原則として 設計はしない = サブタスクはない(github issueが紐付いていても見積もりとしては無視) あくまでユーザーストーリーとして見積もる
        - タスクの流れはあってもよい
        - ただし知らないと見積もりできない場合は「知ることや共有できる」のユーザーストーリーを別に切って進める
    b. エンジニアだけでなくデザイナーやPdMの工数も明確にしておく
        - そこで他のロールとコミュニケーションや作業など発生する場合もあるため
5. 複数パターンの見積もりは避ける
    - 見積もり工数が大きくなるのと、結局漏れが発生するため
6. 他人の作業でも自分達が円滑に進めるかの観点で全員が全てのユーザーストーリーに合意できる状態を作る
7. 見積もり会で完了条件含めて確認し流れに問題がなく全員の見積もり感覚が合うまで膝を付き合わせる
    - 膝を突き合わせる=双方や全員の各ユーザーストーリーに対する熱量が引き出せる状態
8. どうしてもコストや見積もりの感覚が合わない時は、全員の感覚が合うサイズにユーザーストーリーを切り分けて価値とコストの比較を行いやすくする
9. 当該エピック以外の今後の改修など、今見えていない物や、日常の保守運用コストについてもわかる範囲で合意する

見積り時は毎回、上記を確認しながら実施

KAKEHASHI

見積もりを一緒に進めたメンバーの感想

フロントエンドエンジニア

エンジニア以外の方とも認識を合わせができてとてもありがたいです。
リファクタリングなど表に出にくい開発がどんな時に必要になるかなど認識が合うようになってきたと感じます。
必要なことはその場で議論できるので、見積もりに向けた事前準備も最小限で済み、コストも下がったと感じます。

プロダクトマネージャーとして要求定義を完璧なものにせねば！と肩ひじ張っていたこともあったが。
今の進め方はプロダクトマネージャーとして要求定義する責務はありつつも、一緒に点検していってもらおうという気持ちで臨めるのでありがたいと思います。

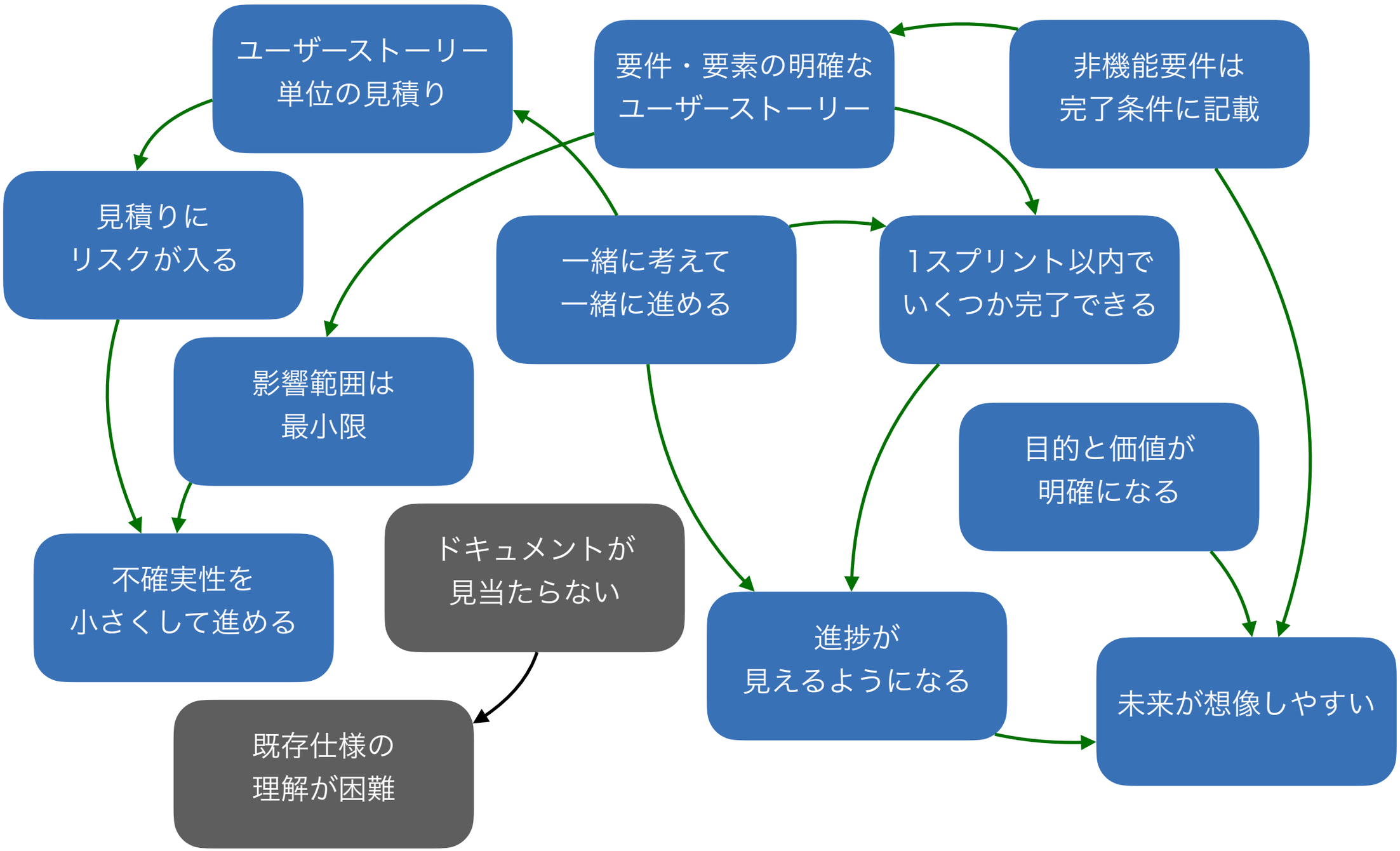
プロダクトマネージャー

バックエンドエンジニア

この考え方で見積もりを始めたことで、少しずつチームの一体感を感じられるようになってきた気がしています。
今は全体の価値提供のプロセスのほんの一部の見積もりだけですがプロダクトマネージャー、デザイナー、ディレクター、エンジニアのそれぞれが同じ共通認識の元で価値が生まれて利用者に届けられるところまでを一緒にやれるチームになれたらいいなと思っています。

KAKEHASHI

徐々に望ましい現状を増やしていく



解決しない部分は他との関連が薄まり別の施策を実施可能に

KAKEHASHI

工夫したポイント

KAKEHASHI

ルールで縛らない

ルールを守ることを目標にするのではなく
自律的な取り組みの中でチームが柔軟に変化する事で
魅力的なプロダクトを継続的にリリースでき事を優先する

前提

- 決まったルールに従うのではなく、目的に沿ってメンバーが動きやすくなる方法を考える
- 常に**完璧であることを目指さず持続可能な運用**を心がける
- 練習と振り返りを通して小さなコストでできている割合を上げたり、精度を上げることを目指す

ルールがないと個人の判断と裁量に任せることになるため
丁寧な推進が必要になる

KAKEHASHI

実施初期からモビングで一緒に記載

PBIを一緒に作ることで
チームと組織が価値を理解しやすくする会

立場やロールに関係なく混合したグループ分けによる開催

プロダクト開発チーム全員が参加可能

エンジニアリングマネージャーは全ての会に参加

記載方法や考え方に慣れていない人が多い中
指示だけして放置するのではなく
記載内容をリアルタイムでフィードバックしたり
現場の難しい問題を一緒に考えて一緒に解決する

KAKEHASHI

記載も検討も見積りもモビングで実施

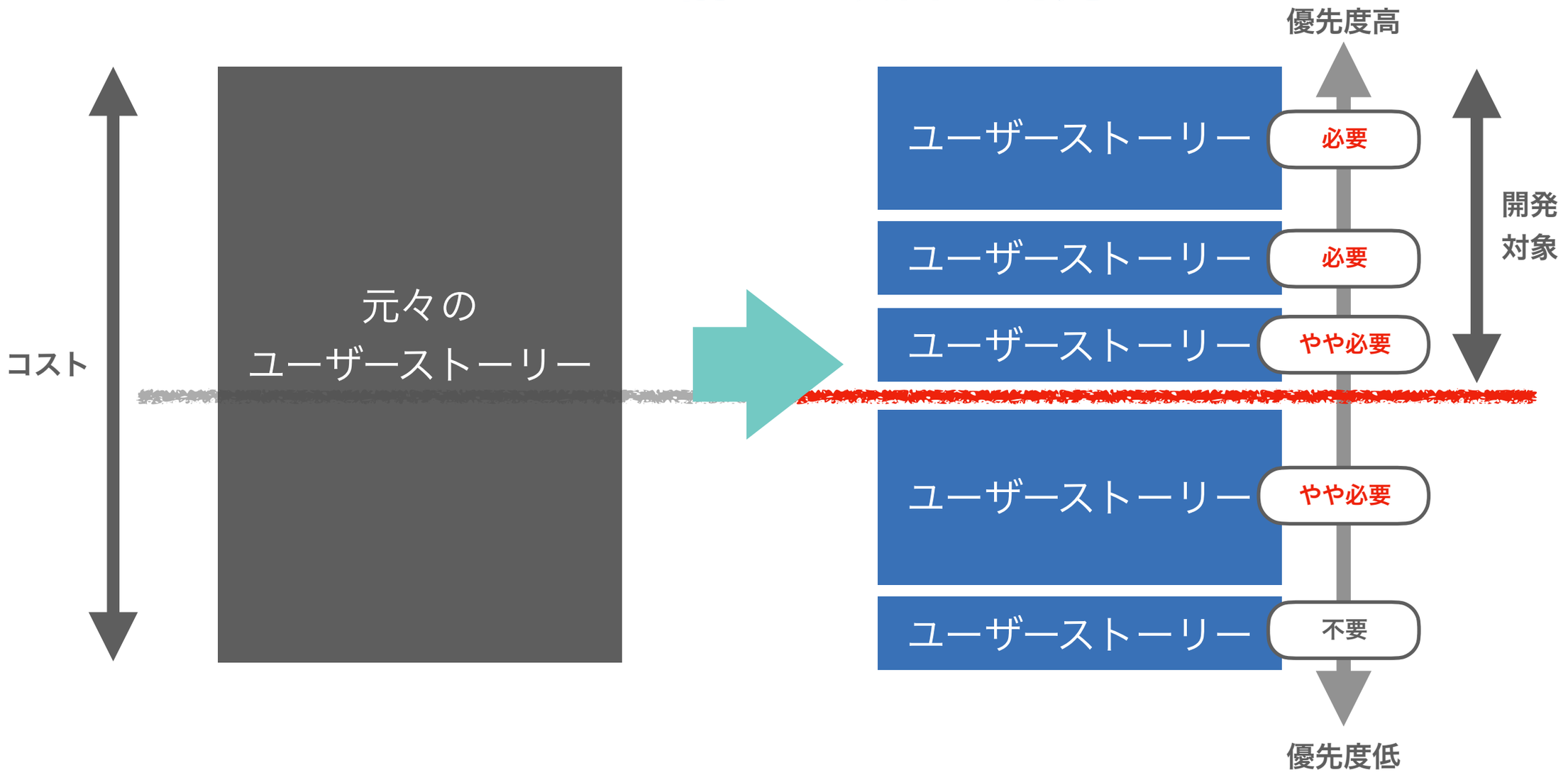
常に一緒に考えながら進めることで
見直しをより多く実施して早期に成功体験を得る

価値が明確になるまで何度も一緒に書き直し価値分割することで
ユーザーセグメントも細かく理解できるように変化

KAKEHASHI

必要のないユーザーストーリーを探す

ユーザーストーリー小さくを明確にすると
スコープとして削れる項目が発見できる

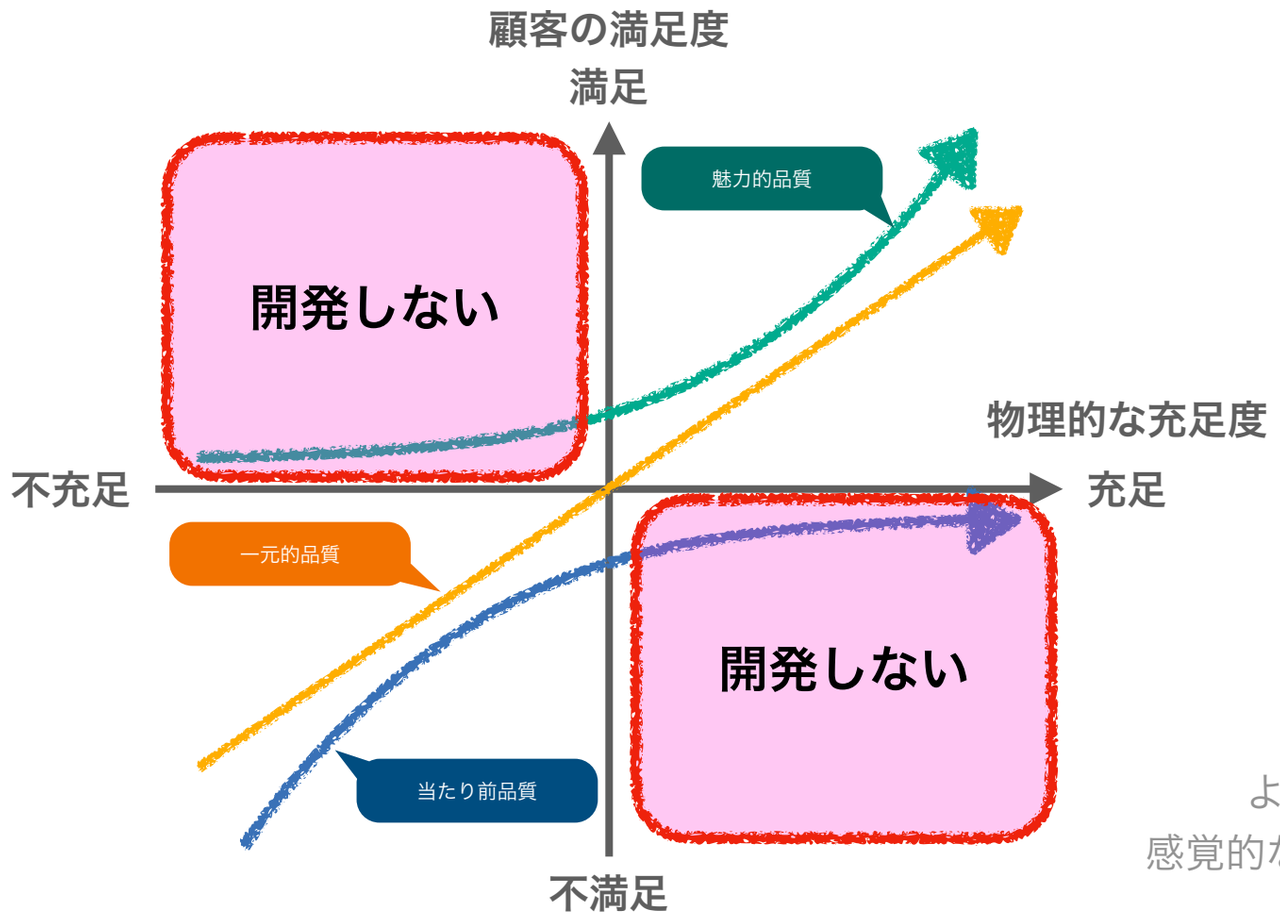


優先度やコストを判断して開発を決定する

KAKEHASHI

開発対象から外したいユーザーストーリー

あって当たり前に思える何かや、ちょっと魅力的に思える何かも
今提供できる体験や利益が高くないなら開発しない



よくある画像に
感覚的な印象をマッピング

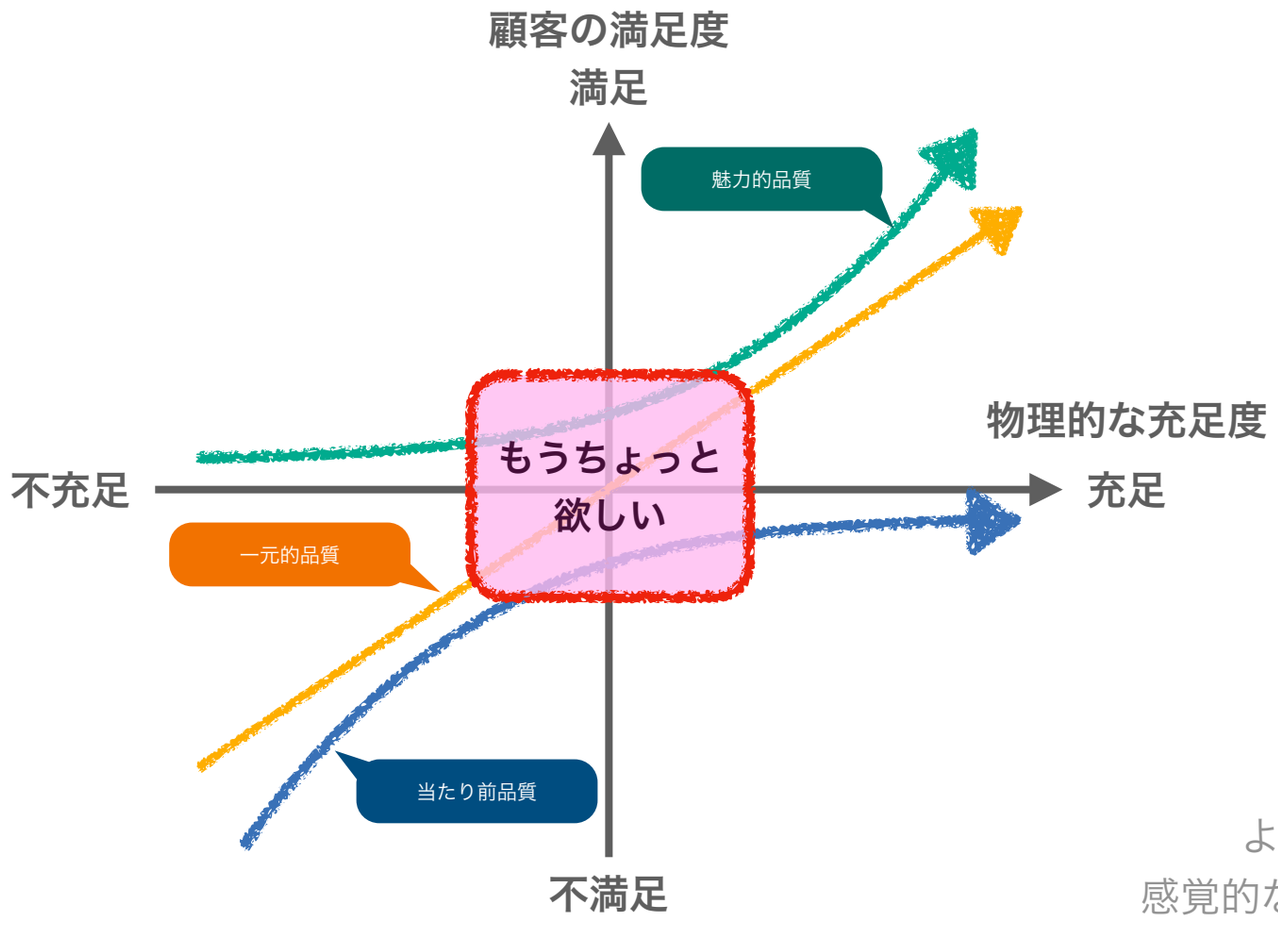
今の状況と今のリソースで提供できる最大の体験や利益を考える

KAKEHASHI

開発対象から外して得られる効果

お客様とCSとの関係性に基づくサービス改善までを提供サービスと捉え

お客様の『もうちょっと欲しい』を上手に引き出して
サービス成長の原動力にする



よくある画像に
感覚的な印象をマッピング

お客様の利用があれば改善要望もらえるプロダクト作りを目指す

KAKEHASHI

実施結果

実施対象期間は2023年5月1日から2023年7月31日の3ヶ月
比較対象期間は2023年2月1日から2023年4月30日の3ヶ月

KAKEHASHI

スケジュールに対する進捗の変化

5月以降のスケジュール遅延は最大１週間程度にとどまり
その後4週間以内にリカバリできている

KAKEHASHI

障害発生の変化

- 全障害件数は2件増加に対し、期間外の実装影響も2件増加のため
  対象期間の作業による障害件数は両期間とも4件

要求の合意漏れによる障害は3件から0件に

期間	障害	高い障害レベル	原因				
			要求の合意漏れ	影響範囲の想定漏れ	インフラ側	初期データ不整合	期間外の実装
2023/2/1 ～ 2023/4/30	5	4	3	0	1	0	1
2023/5/1 ～ 2023/7/31	7	1	0	3	0	1	3

※ 期間外の実装は全て2023年1月以前の実装起因によるもの

KAKEHASHI

期間ごとの障害の内訳

期間外の実装起因を除いた障害のうち
お客様の業務に回避不可能な影響が発生した
高い障害レベルは0件に

期間	発生日	内容	障害レベル	原因
2023/2/1 〜 2023/4/30	2023/2/10	おすすめリストが正しく表示されない	**高い**	要求の合意漏れ
	2023/2/26	エラーにより後続の発注おすすめの計算処理が実行されない	低い	AWS内のエラー
	2023/4/4	合計数量を算出するコンポーネント内のエラーでアプリが落ちる	低い	要求の合意漏れ
	2023/4/18	入庫原価が空欄で登録さてしまう	**高い**	要求の合意漏れ
	2023/4/21	在庫調整を利用した際に正しくデータが送信されていない	低い	期間外の実装
2023/5/1 〜 2023/7/31	2023/5/10	メールアドレスのドメイン部に『-』があるとログインできない	**高い**	期間外の実装
	2023/5/18	画面の未採用薬を表示する部分がスケルトン表示になる	低い	影響範囲の想定漏れ
	2023/5/23	安心買いの数値が正しいく表示されない	低い	影響範囲の想定漏れ
	2023/5/23	タブ間での移動で先に開いた情報が移動先のタブで表示	低い	期間外の実装
	2023/5/24	集計データの不整合により数値が正しく表示されない。	低い	影響範囲の想定漏れ
	2023/6/1	在庫不整合により安心買いの機能が正しく提供できない	低い	初期データ不整合
	2023/7/20	DB のコネクション切断が発生する	低い	期間外の実装

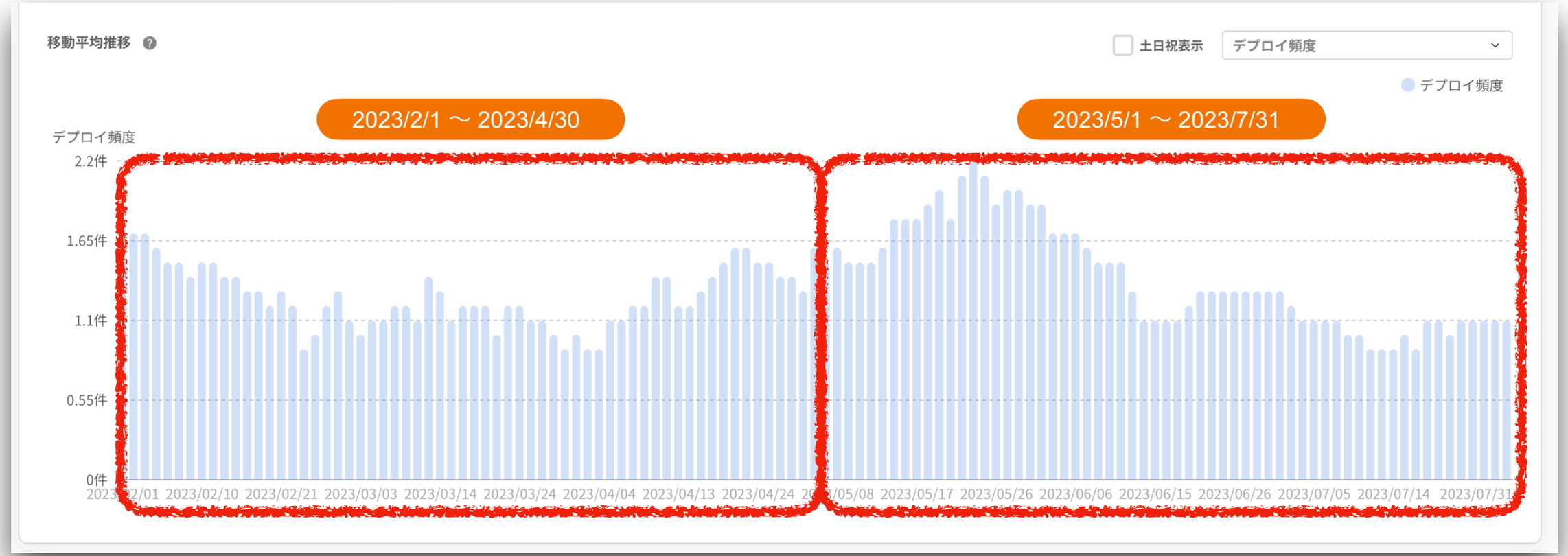
※ 期間外の実装は全て2023年1月以前の実装起因によるもの

KAKEHASHI

期間中のデプロイ頻度の比較

高い障害レベルの障害が低減したのに対し
デプロイ頻度は両期間1回/1日以上程度で推移しており
リリースは安定して継続

Findy Team+



KAKEHASHI

まとめ

実施結果からの考察

- プロダクトバックログアイテムの見直しは品質の向上に繋がったのか？
  - 障害要求の合意漏れによる障害や、高い障害レベルの障害も低減した
  - 必要な開発は進んでおりデプロイ頻度も一定範囲を保っている
  - スケジュールが安定した

品質は向上したと考えられる

- 品質が向上した理由として考えられる要因は何か？
  - ユーザーストーリーの価値が明確になり目的を理解して開発を進めることができた
  - ユーザーストーリー完了時の状態が明確になりメンバー間の認識のズレが減った
  - ロールを跨いで一緒に考えることで各ロール間の理解が深まった
  - 不必要な開発を減らすことで仕様の複雑性が上がらなかった可能性がある

KAKEHASHI

QAチームを作った場合と何が違うのか？

QAチームを立ち上げたら
学習コストや認知・作業負荷を増やすことなく
品質向上ができるのではないか？

- 学習コストや認知・作業負荷は増えたが実際のコスト増加は大きくないと考えられる
  - 実施以前よりスケジュールが安定している
  - デプロイ頻度も維持されている
- 既存メンバーが自律的に行動し変化に対応する事で学習機会と品質の向上が期待できる
  - 品質についてチーム全体で考えることができる
  - 自分達で考え実践する中で自分達で出来ること、出来ないことを知ることができる
- 限られた予算やリソースの中で改善を進めることができる
  - QAチームのための人員を採用したり外部リソースに頼る必要がなくなる
  - 人やチーム増加によるコミュニケーションコストの増加を抑えることができる

KAKEHASHI

わかったこと

今の自分達で出来る事・できない事を知った上で
将来の組織設計を明確にしながら
限られた予算でプロダクト開発チームを運営できる

本当に必要になった時に
QAチームを作れば良いと考えることができる

KAKEHASHI

今後の取り組み

引き続きプロダクトバックログアイテムの改善を実施

特に価値分割を中心とした改善を行い
サービス全体のフィードバックサイクルの高速化を目指します

KAKEHASHI