Sustainable Quality Goalsを追求する人たち

2021.07.16

Sustainable Quality（持続可能な品質）

昨今、頻繁に聞くようになったSDGs「Sustainable Development Goals（持続可能な開発目標）」

ソフトウェア開発においても、製品を売ったら終わり、納品したら終わりという時代から、インターネット経由でサービスを提供し、継続的に利用してもらうビジネスモデルへと変わってきており、持続可能な開発が求められてきています。また、インターネットを取り巻く環境の変化により、関わるヒト、モノ、カネ、情報などが大きく拡大してきています。

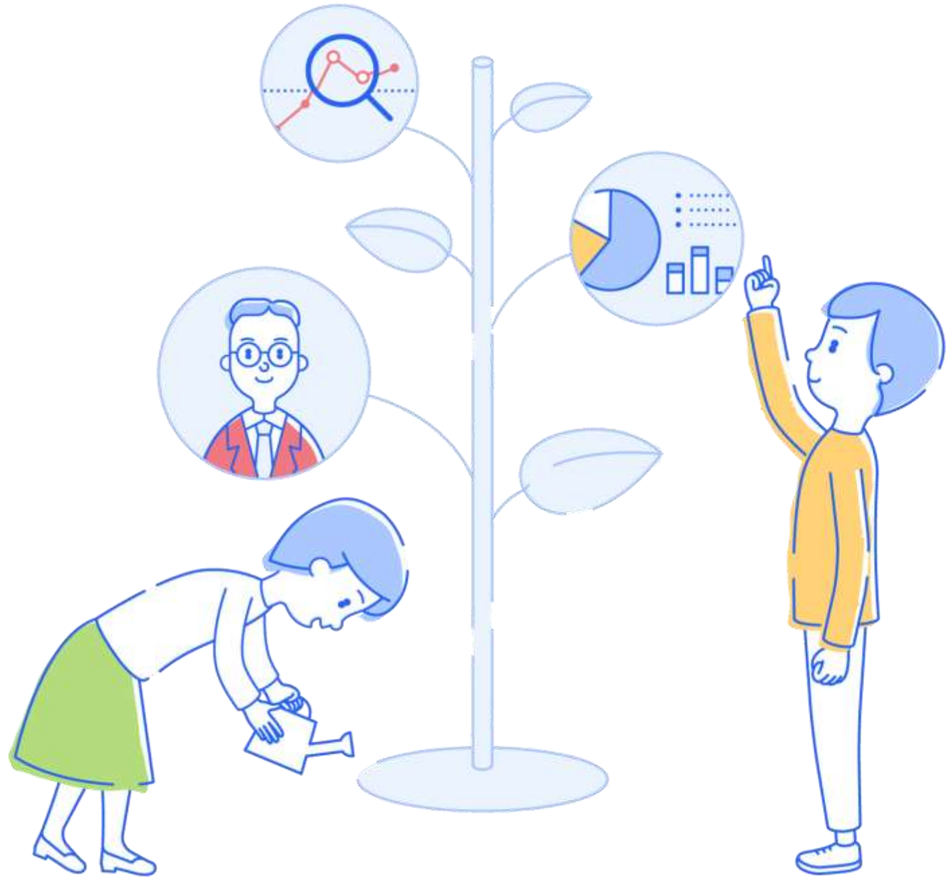
このような変化に対して、freeeでは、どのようなチーム体制で、どのように品質を担保しているのか

今回は、2020年4月20日に新規サービスとしてリリースした「freee プロジェクト管理」というサービスの事例を交えて紹介していきます。

ソフトウェアの変革

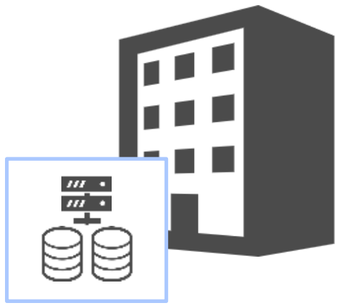
サービスの継続性が求められる

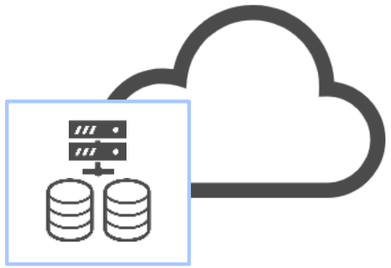
・買い切りモデルからサブスクリプションモデルへ
・継続的にサービスの価値を提供していかなければいけない
・継続的に品質を維持していく必要がある

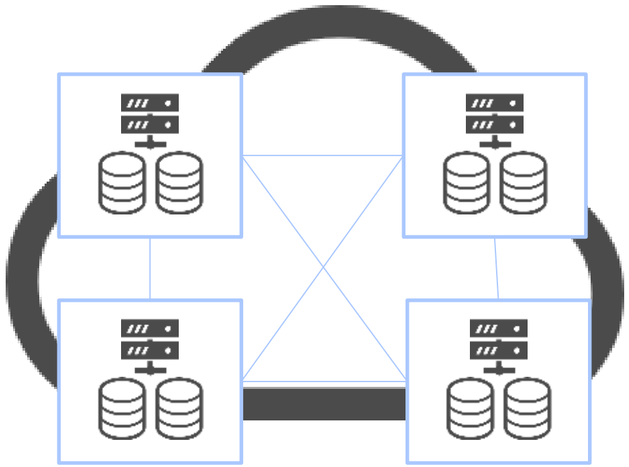


サービスの構成が複雑化

・オンプレミスからクラウドへ
・モノリシックからマイクロサービスへ
・オープンソースを利用したフレームワークやライブラリの組み合わせで構成

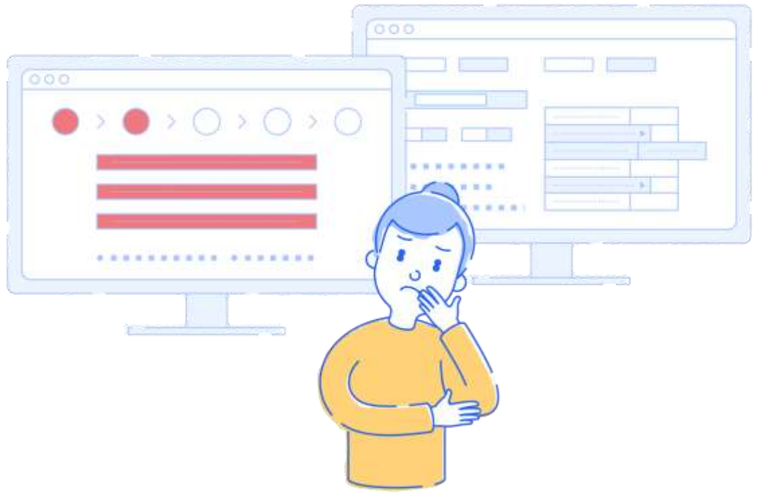






オンプレミス
クラウド
モノリシック
マイクロサービス

サービスの継続性 x 複雑性



見えてきた課題

連携しているサービスの仕様変更によって、サービスが使えなくなった

ライブラリを更新したらサービスが動かなくなった

利用しているライブラリのサポートが終わり、サービスが停止した

以前は動いていた機能がいつの間にか動かなくなってしまった

顧客からの問い合わせで、新機能のリリースを知った

顧客の要件にあわないので、サービスが売れない

画面からできることが、Public APIからはできない

関連サービスが停止して、使えなくなった

問い合わせが多い

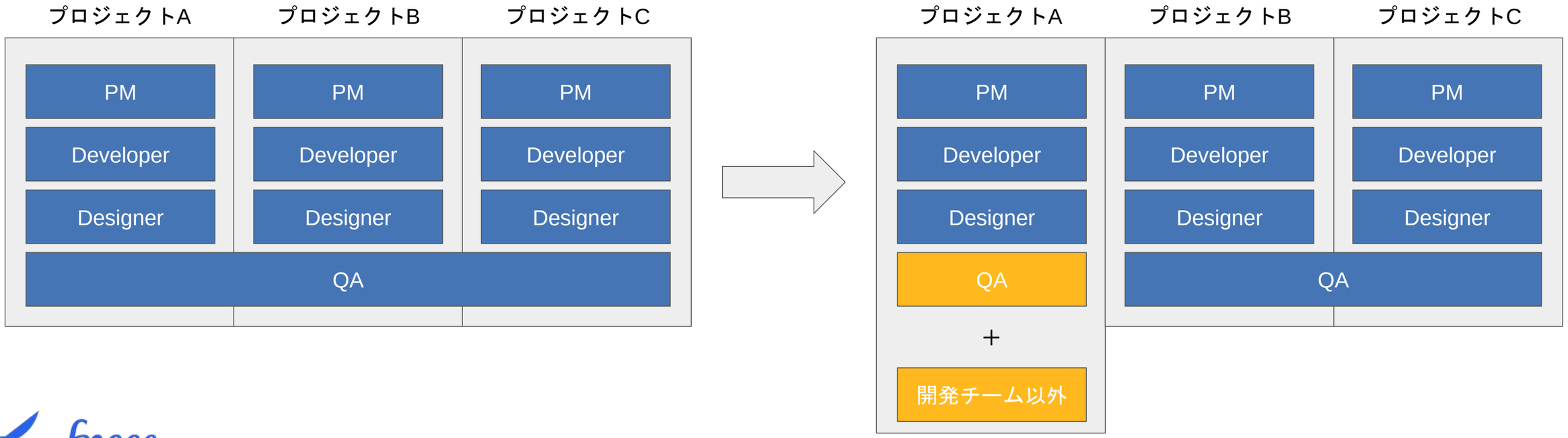
リリース頻度が低下

以前に比べて、反応が悪くなってきた

チーム体制とコミュニケーション

ONE TEAM

・アジャイル開発におけるQAを推進していけるチーム体制作り
・QAメンバーを固定化し、プロジェクトにアサイン
　・仕様のキャッチアップやリスクの提言
　・各種ミーティングへの参加
　・開発エンジニアとのコミュニケーション改善
・開発チーム以外のメンバーも巻き込んでいく



登場人物

PM
要件や仕様を決めるプロジェクトマネージャー

Developer
サービスを開発するエンジニア。関連するサービスの開発エンジニアも含む

Designer
UI/UXを開発するデザイナー

QA
品質保証担当

SRE
インフラ周りを担当するエンジニア

Support
顧客対応を行うカスタマーサポート

Biz/Success
フィールドセールスやカスタマサクセスを行う営業担当

Analysts
データ分析を専門に行うエンジニア

デモは関係者全員でみる

- スプリントレビューでは、そのスプリント内で実装した機能をエンジニアがデモ
- デモには関係者が全員参加
- 今、作っている機能がユーザが求めているものなのかを確認
  - 営業視点からのフィードバックは、開発チームにとって非常に有益
- 事前に機能を確認することで、各チームはその後の準備ができる
  - デザイナー：UIやUXの確認
  - 営業：セールスプランの検討
  - SRE：インフラ周りの懸念点がないかの確認
  - サポート：サポート方針の検討やヘルプページの準備
  - QA：テスト方針の具現化

PM

Developer

Designer

QA

SRE

Support

Analysts

Biz/Success

スプリントレビューでライブリリース
- Bizやサポートの人たちは、リリースの現場に立ち会うことはほとんどないため、今、自分が見ている画面がリアルタイムで変わると盛り上がる。

気軽に話す

- メインはSlackによるコミュニケーション
    - やり取りが2往復続くようなら、Google Meetでつないで話す
- カジュアルなコミュニケーションを重視
    - 特にリモート時は、気軽に話ができることは大事
    - 朝会の10分前は雑談タイム
- リモート飲みも月1ペースで実施
- 可能ならリアルなコミュニケーションも

PM

Developer

Designer

QA

SRE

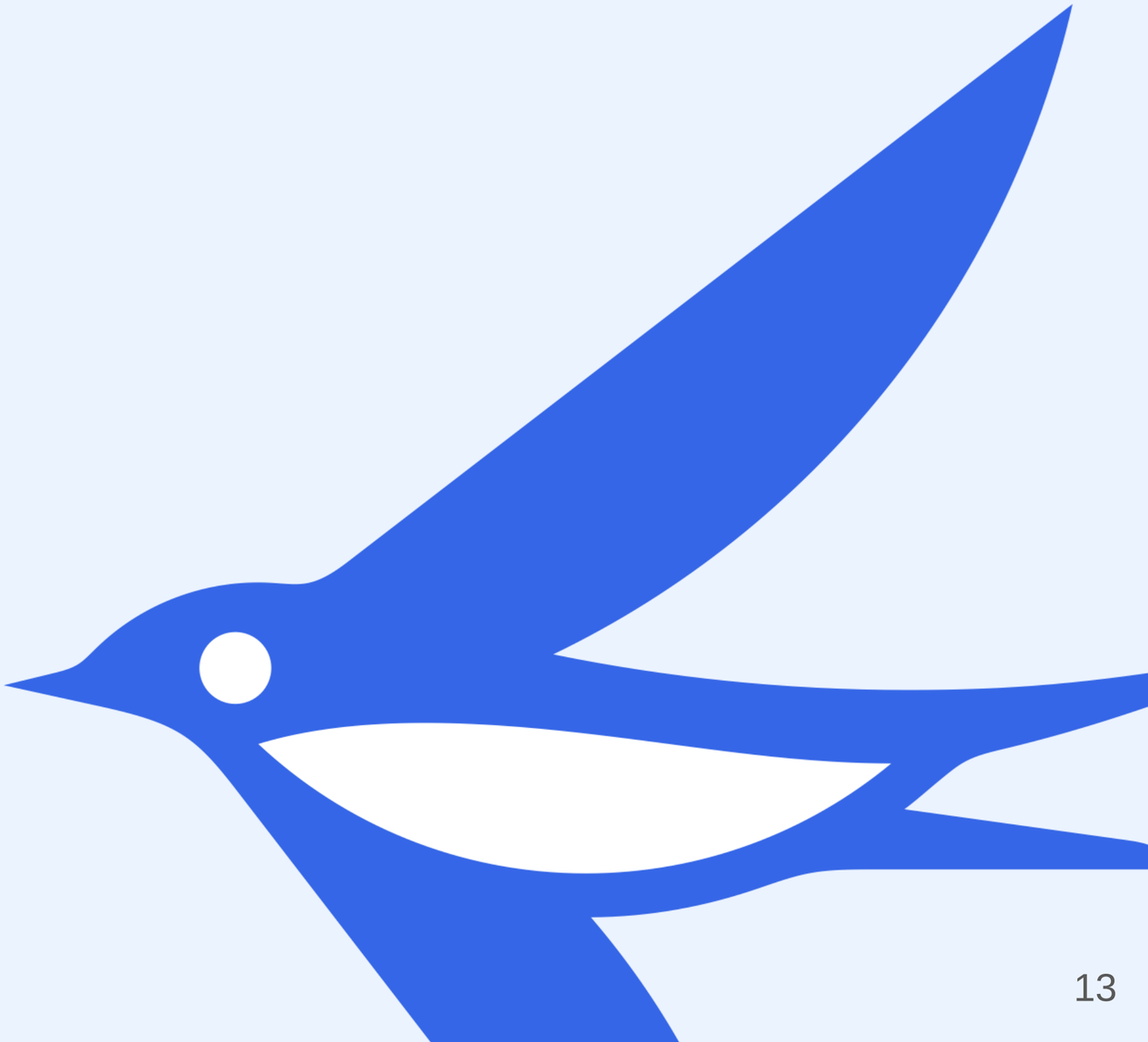
Support

Analysts

Biz/Success

💡コロナ前は、月1回の頻度で大阪へ出張
- 同じオフィスで一緒に作業したり、一緒に食事することで、その後のコミュニケーションが格段に良くなった
- 「同じ釜の飯を食う」のは大事

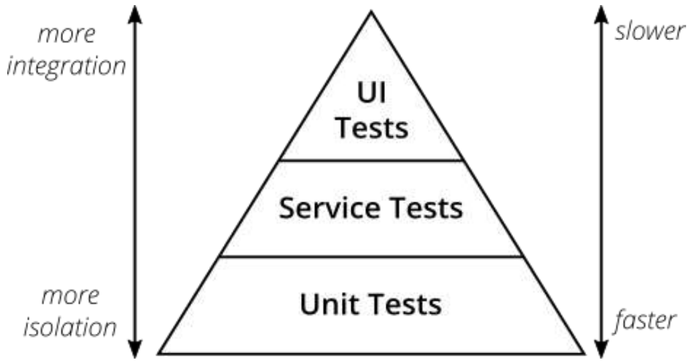
持続可能な品質



テストのカバレッジを維持する

機能が増えてくると、実装による影響範囲が広がり、想定していなかった部分で不具合が発生するケースも増えてくる

- 単体テストは、他のテストに比べて早いし、コストも低い
  - 高いカバレッジを維持することで、品質の低下を抑える
- C0カバレッジは、90%以上を維持
  - Working Agreementに定義しておく
  - カバレッジの目標をOKRに含める
- 定期的にカバレッジの数値を確認
    - データをとるだけではダメ
  - 監視してアクションにつながることが大事



出典: https://martinfowler.com/articles/practical-test-pyramid.html

💡 レトロスペクティブでは、毎回、カバレッジを確認して報告
- 前回から「1%上がった」、「0.5%下がった」、「このモジュールが下がっている」といった情報をQAから報告している
- 現在の状況がわかると、エンジニアのモチベーションアップにつなげられる

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

E2Eテストのメンテナンス性をあげる

機能実装によりE2Eテストが動かなくなる。メンテナンスができなくて、利用されなくなる

- 単体テストに比べるとコストは高いが、継続的にリリースを行っていくうえでは欠かせない
- 細かく作りすぎない
  - 基本となるシナリオをカバー
  - 連携サービスとの疎通確認
- 壊れにくいように作る
  - PageObjectパターンで実装
  - ページ変更による影響を減らすために、各要素にIDやdata-test属性を付与
- エンジニアは、実装後にE2Eがパスすることを確認
  - 失敗する場合は、エンジニア自身がE2Eを修正

💡最初は、QA側でE2Eの実装をしていた
- E2Eが失敗した際に原因となるPRオーナーに対して注意喚起をしているうちに、エンジニアもちゃんとメンテナンスしていく機運が高まってきた
- IDやdata-test属性を開発エンジニアが自ら付与してテストを書くようになったため、メンテナンス性が飛躍的に向上

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

パフォーマンスの劣化を防ぐ

データ量の増加や不正なコードの混入によって、パフォーマンスが徐々に悪くなってくる

- リリース前に必要最低限の性能テストは実施
  - QA/Dev/SREが三位一体で実施
  - 最大値の見積もりが難しいし、サーバー利用のコストも大きい
- リリース後は、利用状況を監視しながら、追加でテストを実施
  - APIの応答速度やデータ量を監視
  - 利用者の状況をデータで確認
  - 営業状況をヒアリング

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

Bizチームの営業状況を把握しておく

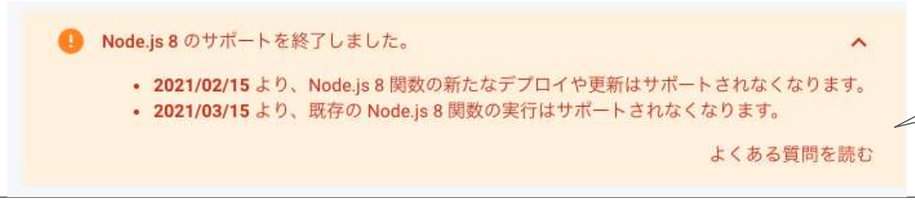
- 「今、某企業に売り込みをかけていて、うまくいけば3,000人ぐらいが使ってくれそう！」（営業）
- 「3,000人を想定したテストって、まだやったことないので、事前に確認しておいたほうがいいね。」（開発チーム）

- 「今回受注した会社は、毎月100件ぐらいの案件があるそうです。」（営業）
- 「毎月100件ということは、年間1200件、5年で6,000件ぐらいの案件数が想定されるので、今の性能だと処理できなくなりそう。今のうちにチューニングを進めておこう。」（開発チーム）

外部モジュールの更新は慎重に

外部モジュールの更新やサポート終了により、サービスが停止したり、一部の機能が正常に動作しなくなる

- ミドルウェア、ライブラリ、フレームワークといった外部モジュールは、安易に更新しない
  - 破壊的な変更が含まれていないかをきちんと評価する
  - 差分が大きい場合は、十分にテストしてから更新
  - パフォーマンスの劣化がないか監視
- サポート期限の把握
  - 使っている外部モジュールをリストアップしておき、定期的に確認
  - 期限前にサポートを切られる場合もあるので、余裕を持って対応



2020年6月25日にサポートが突然切られて、広範囲にわたって障害が発生！

外部モジュールの更新は、Tech Leadのレビューを必須としている
- 変更内容に応じて、対応方法をあらかじめ決めておく
  - メジャーアップデートや破壊的変更を含む場合は、サービス全体をテストしてから
  - マイナーアップデートや変更が小さい場合は、E2Eやブラウザチェックを実施する

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

上流工程で品質を担保する

QAで不具合が多く発見されたり、手戻りが発生したりすることで、リリースが遅くなる

- エンジニアが実装開始と同時にQAはテスト設計を行う
  - 実装中にテスト設計のレビューを実施
- QA中に致命的な不具合や手戻りが発生することを防止
  - 致命的な不具合の発見率は5%以内（他サービス：12〜25%）
- 不具合が残っていてもリリースする
  - トリアージを行い、すぐに影響がない不具合は後で修正

出典：ProductZine
プロダクトの品質はテストだけでは測れない——新規プロダクト開発における品質管理の考え方

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

レビュー会を行うことで、不具合のタネになりそうな部分を除去していく
- 仕様の確認もしますが、QAとしては「こんなテストしますよ」「ここ大丈夫ですよね？」というスタンスでフィードバック
- 「あ、ここバグになる」、「ここ、認識間違ってた」など、仕様の不備や実装漏れが見つかることも多い
- エンジニアは実装中に対応ができるので、QA後に修正するより圧倒的に早いし、コストも最小に抑えられる

連携サービスの変化に気づく

連携しているサービスの仕様変更により、担当サービスが動かなくなる

- 自動テストによる外形監視
    - APIテストによるAPIの変更を検知
    - E2Eテストによる UI周りの変更や仕様変更を検知
- 別サービスと連携する機能を開発する場合は、連携先のメンバーも巻き込む
    - 要件定義や仕様検討のレビューを一緒にやる
    - テストケースのレビューを一緒にやる
- 各ファンクションでの横のつながりと情報共有が大事

PM

Developer

Designer

QA

SRE

Support

Analysts

Biz/Success

会計連携を開発中のエピソード

- 同時期に会計チームは、経費の赤伝対応を行っていた
- プロジェクト管理は、マイナスの経費は許容しない仕様だった
- レビュー中に発覚して、急遽、仕様を変更して対応

品質の低下を見逃さない

以前は、動いていた機能がいつの間にか動かなくなっている。以前と動きが変わっている。

- 問い合わせ対応やリファクタリングが原因でバギーなコードが混入する可能性がある
  - リリース前に全てのPRを確認
  - 影響がありそうな場合は、テストを実施
- リリース時に対応しなかった不具合を放置しない
  - リリース後、2スプリント内に対応
    - 実装からの期間が長くなればなるほど、修正のコストは高くなる
  - 朝会で不具合の対応状況を確認
  - 新規開発をしない改善用のスプリントを途中でいれる
- 品質をみるためのKPIとしてDRE（欠陥除去率）を計測

  DRE ＝（社内で発見した不具合）/ (社内で発見した不具合＋社外で発見された不具合）

新機能をリリースした後、エンジニアとQAで振り返りを行っている
- 不具合の要因分析と再発防止策を検討
- リリース時に見送った不具合が残っている場合は指摘

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

ユーザからの問い合わせを減らす

ユーザ問い合わせが増えると、サポート対応に時間をとられ、本来の開発リソースが削られ、Velocityが低下する。リリースが遅れる。

- Dailyの朝会で問い合わせ内容を確認して、トリアージを行う
  - 問い合わせを溜めていかない
  - 日替わりで日直担当者を決めて対応する
- 問い合わせ内容の分析を行い、すぐに対応が難しいものはバックログに積む
  - 通常のバックログとは別に積んでいる（手が空いた時にやるためのバックログ）
- サポートチームへの情報共有

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

💡QA中に発見した不具合で、「仕様通り」として閉じたチケットの情報をサポートチームに共有
- 「仕様通り」だとしても、QAで不具合だと思った事象は、ユーザも同じ事を思う可能性が高い
- 「なぜ、仕様通りなのか」の情報を事前に提供しておくことで、サポート側での対応がスムーズにいくし、開発チームへエスカレーションする手間が省ける
- 問い合わせが予測される場合は、ヘルプに記載したり、チャットボットで返信することで、ユーザの手間も減らせる

ユーザのことを知る

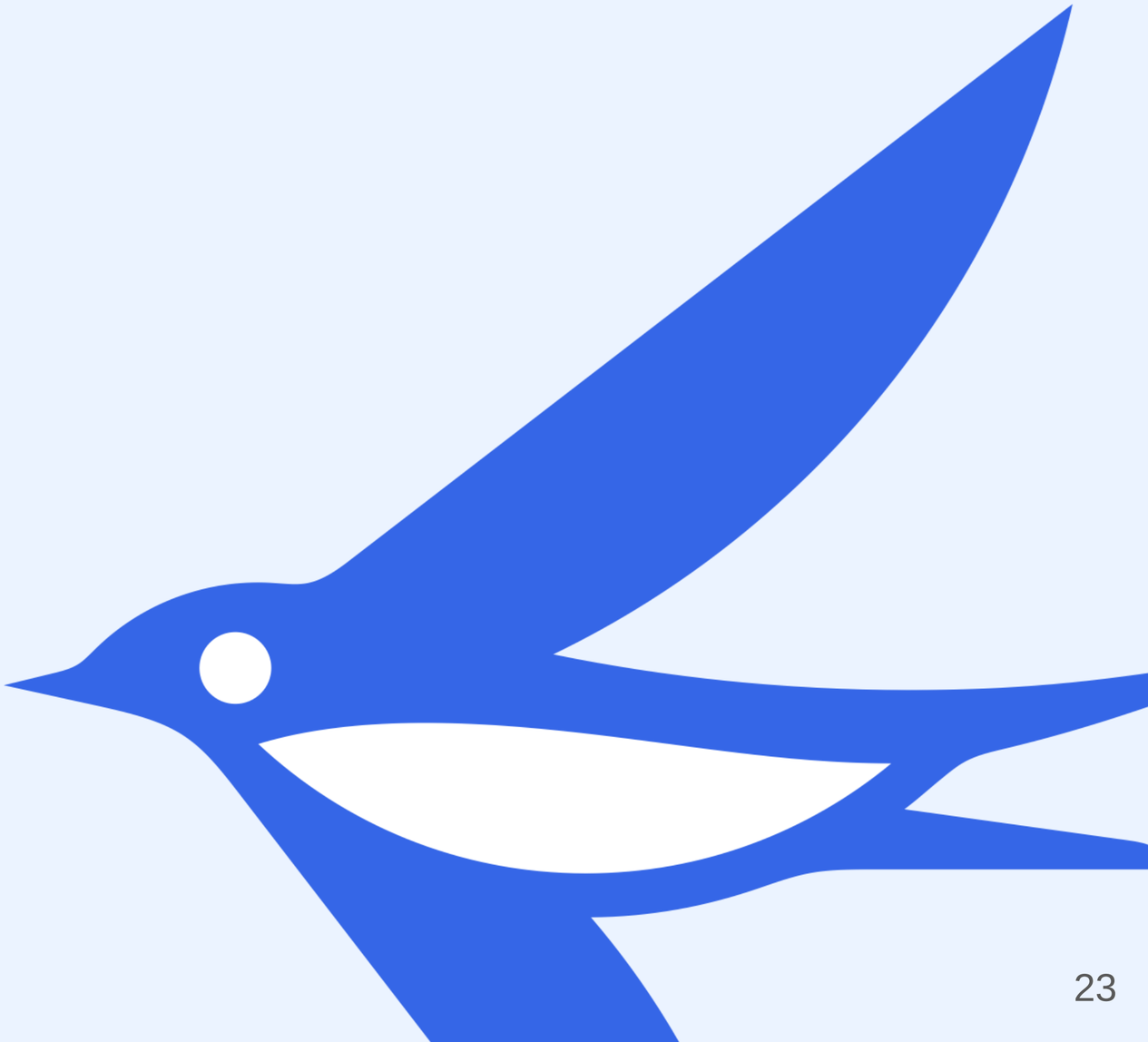
新規機能をリリースしても使われなかったり、ユーザが求めているものとGAPがあり売れない。逆に、ユーザ問い合わせやクレームが増える

- 定期的にBiz担当との開発メンバーでMeetingを実施して、開発及び営業状況を共有
  - 「商談はプロダクト開発の一部」と、とらえる
  - ユーザのペインをヒアリング
  - 開発のロードマップをBiz担当者が知ることで、柔軟な営業が可能
- どういう機能が利用されているか、データから分析
  - テストの優先度を決める

PM
Developer
Designer
QA
SRE
Support
Analysts
Biz/Success

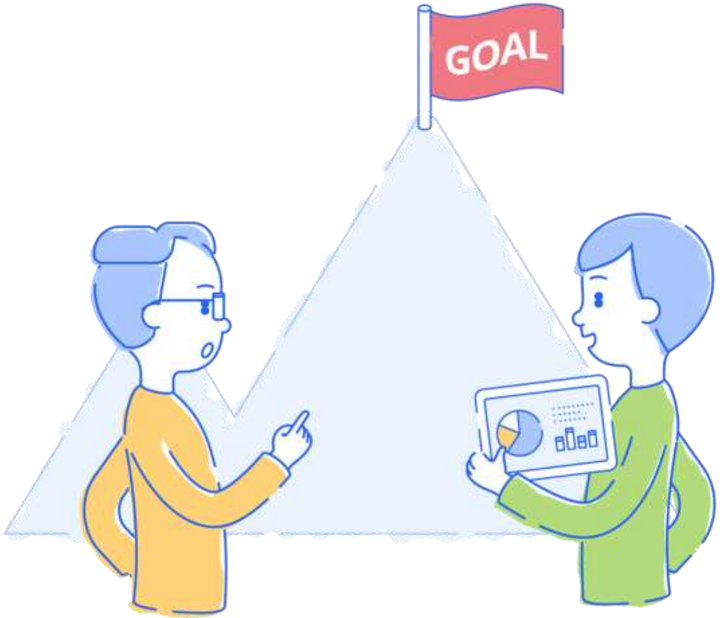
💡ユーザのペインを知ることで、ユーザ目線で評価するためのインプットにしている
- 不具合だけを報告するのではなく、使いにくさやわかりにくい点も指摘
- QA側で想定していなかったテストシナリオを追加

まとめ

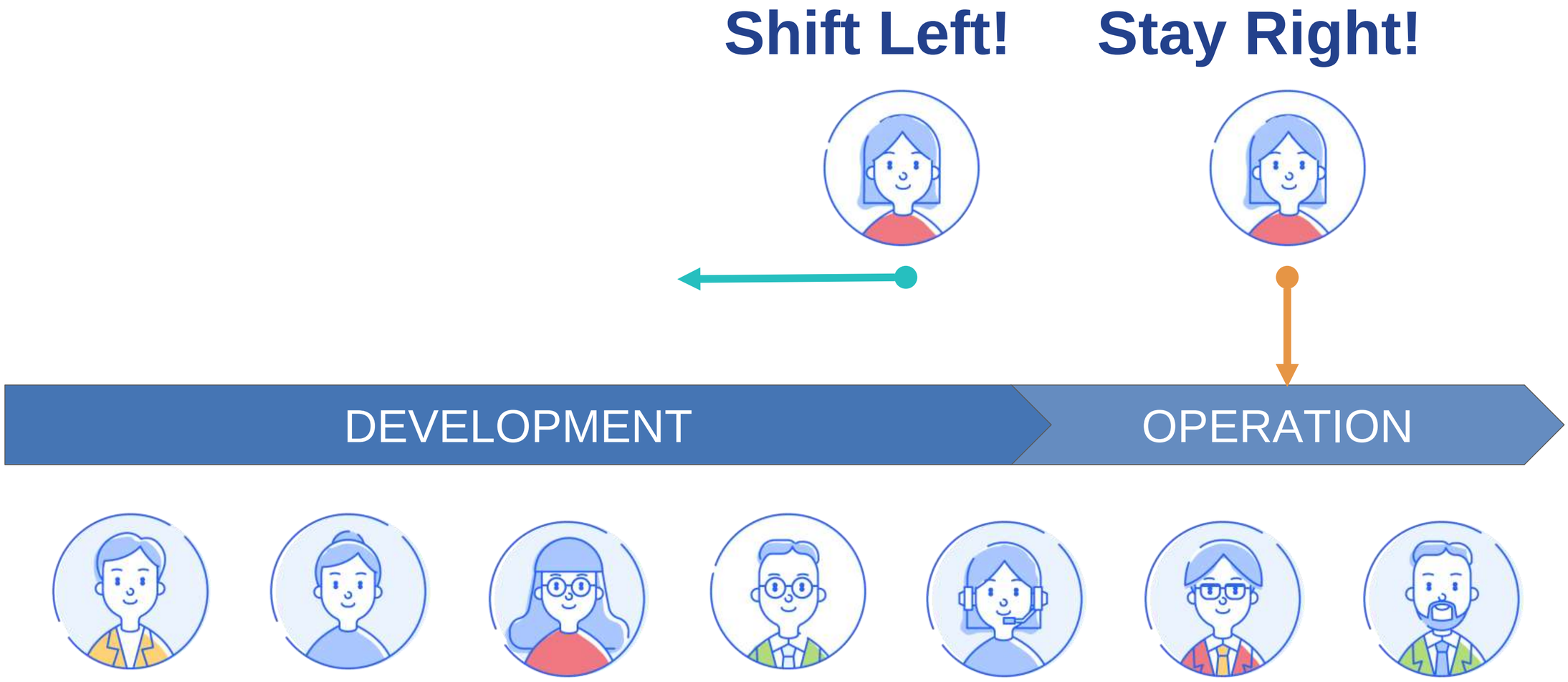


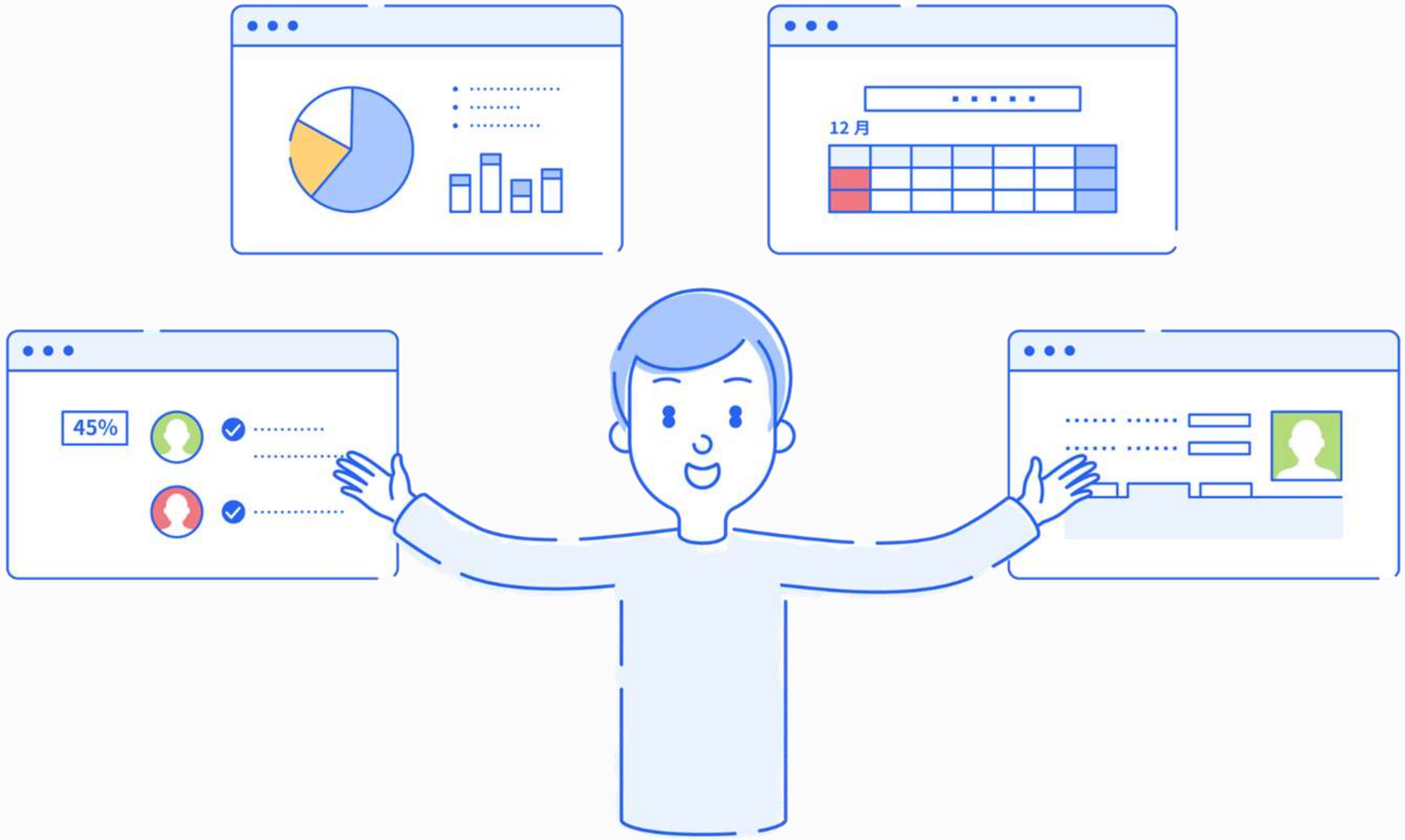
まとめ

- スピードを落とさず、品質も落とさない
  - 上流工程で品質を担保していく
  - 単体テスト・E2Eテストといった自動でテストする仕組み
  - 連携部分に注意
- リリースしてからもテストは続く
  - リリースは、「終わりのはじまり」
  - 負債を貯めていかない
  - 不具合を埋め込まない
  - ユーザの利用状況を把握しながら、継続的にテストしていく
- サービスに関係する全ての人たちを巻き込んでいく



Sustainable Quality Goalに向けて





ご静聴ありがとうございました。