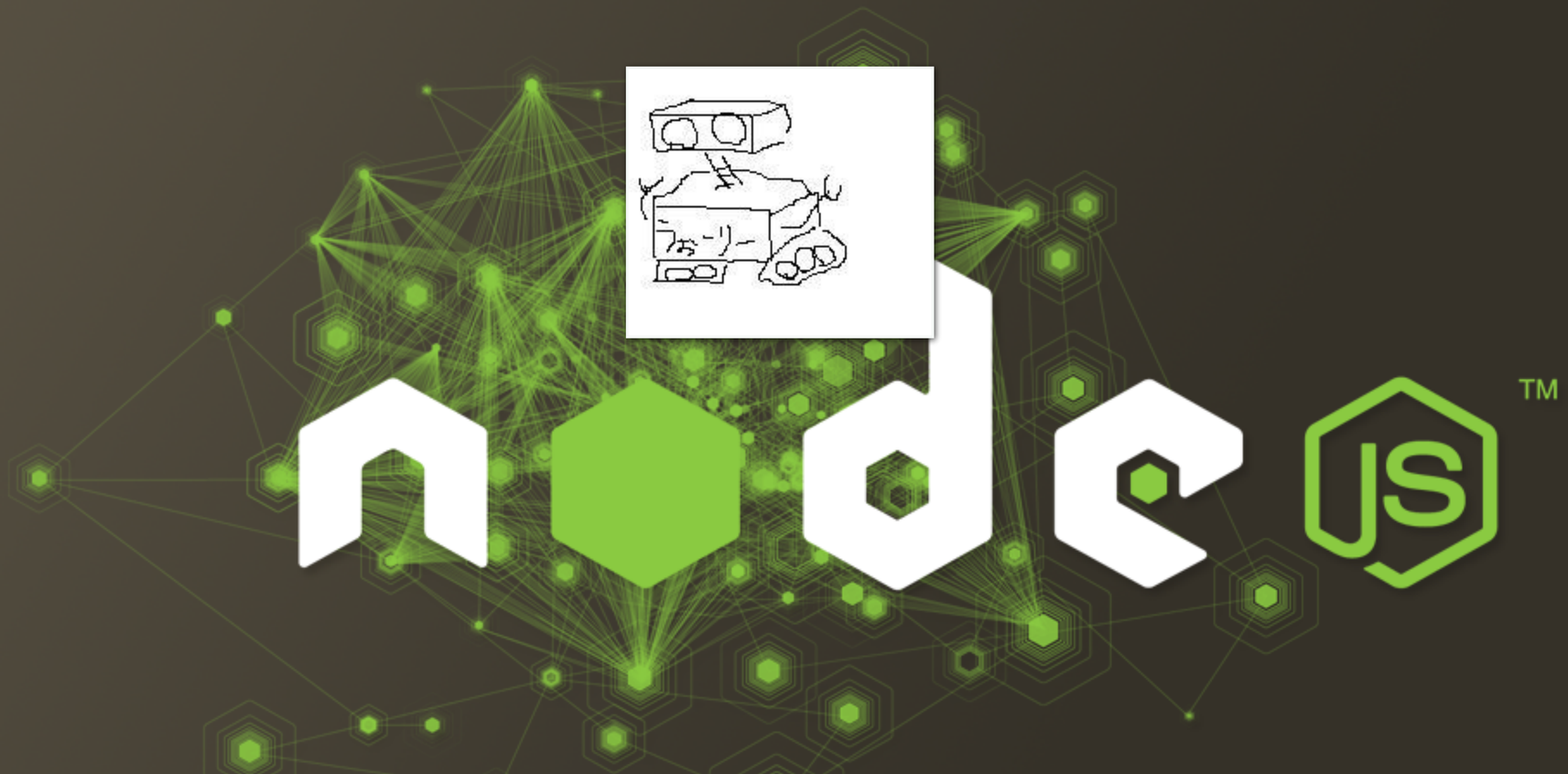
# フロントエンドのDXと今後

6/15 @ Developer Experience Day 2023



Twitter: @yosuke_furukawa

Github: yosuke-furukawa

# 技術顧問を数社している

- リアーキテクトについての話
- テストのやり方の話
- コードレビューのやり方について
- アプリのパフォーマンスを上げたいけど...

# 技術顧問を数社している

- リアーキテクトについての話
- テストのやり方の話
- コードレビューのやり方について
- アプリのパフォーマンスを上げたいけど...

**やりたい開発ができていない**

# 原因は様々

- メンバーの習熟度が足りない
- プロジェクトの時間が足りない
- 非機能要件の重要性を理解してもらえない
- 過去の開発プロセスから抜け出せない

# なんで "それ" がしたいの？

- 開発の生産性を上げたい
- ただこの生産性という言葉自体が多様な意味を含んでいる
- 時間軸が存在していて、そこをどう切るかによって変わる

# 開発の生産性を上げたい

- 上げてどうしたい？
  - もっとアウトプットしてアウトカムを稼ぎたい
  - 仮説 => 検証のプロセスを何度も回したい
- 他には？
  - モダンな開発を学ぶことで技術者として学びを得たい
  - イケてるフレームワークにすることで技術プレゼンスを上げて採用に貢献したい

# 開発の生産性を上げたい

- 上げてどうしたい？

**具体的な価値を提供したい**

  - もっとアウトプットしてアウトカムを稼ぎたい
  - 仮説 => 検証のプロセスを何度も回したい

- 他には？

**あるべき姿にしたい**

  - モダンな開発を学ぶことで技術者として学びを得たい
  - イケてるフレームワークにすることで技術プレゼンスを上げて採用に貢献したい

価値を提供することを value
から来るという意味で valid
な状態を指す

正しくあるべき姿にすること
は very からきており、
verified な状態を指す

# 価値の提供とあるべき姿の追求

- これらはどちらかが欠けていても問題、どちらも達成していく必要がある
- もちろんビジネスという意味では価値の提供の方に重点が置かれる
- 一方であるべき姿ではないと徐々にものづくりから現場のメンバーが離れて行ってしまう

このバランスをどう保つかは
やっぱり難しい

2つのアプローチで
解決していきたい

# 1. ボトムアップ

# ボトムアップのアプローチ

- 現場のレベルで知見をためて上げていく
- 勉強会だったりハッカソンだったりも有効
- 教えてくれる人を捕まえて直接教えて貰う方法もある

# 2. トップダウン

# トップダウンのアプローチ

- 指針となるものを決めてそこからおろしていきたい
- フロントエンドの設計に指針をもたせられていないから方針がブレる
- 設計そのものはアプリケーション特性に寄って変わるが、設計をするための方針ならある程度寄せ集めた知見を元に作ればできるのではないか・・・？
- フロントエンドの指針を作れないか・・・？

そんな折に出会ったのが
こちら

**DX Criteriaとひろきだいちと**
**のパネルトーク**

# パネルトーク

- フロントエンドの組織論の話(postdev)


https://blog.nijibox.jp/article/postdev1/

# ここでの話し合いの内容

- 画一的なアプローチではなく、その場その場に応じたアプローチが取れる組織こそが強い組織
- 強い組織になって開発をする際のヒントとしてDX Criteria のような指針を用意したい
- できればフロントエンドに特化した DX Criteriaとかを検討してみたい

# ここでの話し合いの内容

- 画一的なアプローチではなく、その場その場に応じた
- 強い
- でき

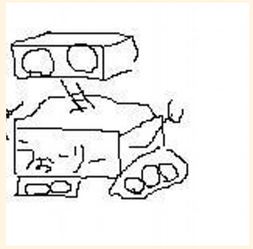
とのことで

# 本題:
# フロントエンドにおける
# DX Criteria を~~作った~~作ろうと
# している話

# フロントエンド版DX Criteria

- 本家 DX Criteria は高速な仮説検証を回して組織内の素早い検証と失敗と学びを最大化しようとするための方針で作られた奴
- そのエッセンス部分は残しつつ、フロントエンド領域に特化する
- 本家は会社の話も多いが、フロントエンドという領域に閉じるので、会社の話は少なめ
- 一方でデザインとの協業や関連部署との話は多め

# フロントエンド版 DX Criteria

- 作成者
  -  yosuke_furukawa
  -  ahomu
- レビュー
  -  hirokidaichi

# フロントエンド版 DX Criteria

- 主要5項目
  - 開発・運用の生産性を支える技術スタック
  - ユーザー体験を支える非機能要求
  - 価値のデリバリーを行うプロセス
  - システム・アプリケーション運用（アーキテクチャ）
  - チームビルディング

ちなみにまだ
磨き込み中です！！！

# 開発・運用の生産性を支える
# 技術スタック

# 開発・運用の生産性を支える技術スタック

- 生産性を上げるために技術面でライブラリやツールを使って対処する
- ただ逆にこれもやり過ぎは注意
- クライアントサイドの利用環境、UIのベストプラクティスの変化、ライブラリのアップデートなど、とにかく変更や修正が多い箇所であり、そこに対してどう取り組むかというテーマ

# 例を挙げる

# 開発・運用の生産性を支える技術スタック

- 静的検査と型付け
  - Linter, formatter, editor, 果ては TypeScript でやってるか、、、など。
  - TSを最初から基準に入れるかは迷ったが、もう殆どの開発者が使っているものとして選定
  - 開発のツールを使うのは良いが、振り回されないようにしたいよね、という思いを込めている。

# 開発・運用の生産性を支える技術スタック

- 静的検査と型付け
  - （学習と改善）静的型付け言語やコードフォーマッターを用いてコードベースの読みやすさと保守性を向上させているか
  - （メトリクス）バクの発生率、コードレビューの時間などのメトリクスを取得し、活用しているか
  - （プラクティス）型情報をAPIレベルで共有するためのツールを活用しているか
  - （アンチパターン） 静的型制約を適切に使用して、未然にバグを防ぐことを優先しているか、あるいは型制約を無視して進捗を急ぐ傾向がないか

# 開発・運用の生産性を支える技術スタック

- ツールチェインと開発効率
  - ツールによる効率性のアップがどこまでできているか
  - 例えば build ツールが遅すぎたりしないか
  - Mock とかを適切に使って開発効率上げてるか

# 開発・運用の生産性を支える技術スタック

- ツールチェインと開発効率
  - （学習と改善）Linter, formatter などツールセットを共通化しているか
  - （メトリクス）ツールの実行時間を計測し、効率性を定期的に計測できているか
  - （アンチパターン）一貫性のないツール選定や過剰なカスタマイズにより、開発効率が低下してしまっていないか

# 開発・運用の生産性を支える技術スタック

- 持続可能な運用のための技術選定
  - 技術トレンドに振り回されていないか
  - アプリケーションの特性は理解しているのか
  - 事業目線でライブラリやフレームワークを選定しているか

# 開発・運用の生産性を支える技術スタック

- 持続可能な運用のための技術選定
  - (学習と改善) 技術トレンドを追跡し、新たなツールやライブラリの評価を定期的に行っているか
  - （メトリクス） 既存の使用技術のアップデート頻度、技術スタックの複雑さ、セキュリティ事象などを評価・改善しているか
  - （アンチパターン）流行りの技術や最新ライブラリを無理に取り入れ、技術スタックが過剰に複雑化してしまっていないか

# ユーザー体験を支える
# 非機能要求

# ユーザー体験を支える非機能要求

- 非機能要求をガン無視して進めると結局出来上がったものが使いにくくなる
- どの非機能要求にどの程度応えるべきで、どう対処するかを定義して、検討材料にしてもらいたい
- フロントエンドで言えばパフォーマンス、アクセシビリティ、メンテナンサビリティ、デザイン、などなどたくさん非機能要求がある

# 例を挙げる

# ユーザー体験を支える非機能要求

- パフォーマンス
  - レイテンシーとスループットの概念を理解し、何が何でも「Lighthouse 100点だ！」とかになっていないこと
  - 適宜パフォーマンスチューニングコンテストなどで学べるタイミングを作って欲しい
  - 現状分析と目標設定ができる材料が揃っていてほしい

# ユーザー体験を支える非機能要求

- パフォーマンス
  - （学習と改善）CPUやメモリ、ネットワークと言った基本的な項目についてのプロファイルを取れ、見方について学習しているか
  - （メトリクス）JavaScriptやCSS、画像といった静的アセットのデータサイズを計測し、パフォーマンスバジェットとして予算管理ができているか
  - （アンチパターン）自分たちが配信しているウェブアプリケーションがどれくらいのファイルサイズを配信しているかを把握していない
  - （アンチパターン）目標を持たずに計測し、際限なくチューニングをしようとしている

# ユーザー体験を支える非機能要求

- アクセシビリティ
  - 割りとまだ導入できている開発が少ない領域
  - 特に開発面ではツールは出てきているものの、本来的な意味でのアクセシブルなアプリケーションの開発はまだトライアルをしている組織が多いという印象

# ユーザー体験を支える非機能要求

- アクセシビリティ
  - （メトリクス）アクセシビリティの静的解析を行うツールを用いて、定期的に計測を行っているか
  - （プラクティス）実装チェックリストを作成し、準拠していることを確認しているか
    - 参考: https://waic.jp/docs/jis2016/test-guidelines/202012/
    - https://www.digital.go.jp/resources/introduction-to-web-accessibility-guidebook/
  - （アンチパターン）HTMLのセマンティクスを無視した実装になっている
  - （アンチパターン）タッチパッドやポインティングデバイスでのみ制御できるUIになってしまっている
  - （アンチパターン）チェックツールだけに頼り、テスト実施者による確認が行われていない

# 価値のデリバリーを行う
# プロセス

# 価値のデリバリーを行うプロセス

- CI/CDなど継続してデリバリーを行う際にも「高速に仮説検証する」仕組みが必要になる
- フロントエンドも例外ではなく、特にA/Bテストなどで作りを違えると問題になりやすい
- 他にもエラーや障害発生時にインフラに任せきりにするなどの対応が取られてしまうと問題になる

# 例を挙げる

# 価値のデリバリーを行うプロセス

- テスト
  - （学習と改善）テストカバレッジ基準や自動テストを用意し、これらを継続的に改善するための工数がチームで取られているか
  - （メトリクス）カバレッジを追加してテストがどこを通過しているのか確認できている
  - （メトリクス）すべてのテストがPRごとに30分以内で終わることを確認している
  - （プラクティス）PRごとに回すテストなのかそれとも定期実行するテストなのかを定義している
  - （プラクティス）unit テスト, integration テスト, e2e テスト, Visual Regression テストなどの各種テストの特性を理解し、適切に運用できている
  - （アンチパターン）一部の人だけがテストを書き、一部の人はテストを書かないといったように自動テストを個々人の努力目標などになっている。

# 価値のデリバリーを行うプロセス

- デプロイ
  - （プラクティス）a/b test, feature flag, canary releaseなどの試して実践した後で変更しやすいデプロイを行っているか
  - （プラクティス）異常時にすぐに切り戻せるように Blue Green Deploy などの仕組みがあるか
  - （アンチパターン）デプロイ一回に対して1時間以上の時間がかかる

# システム、アプリケーション運用（アーキテクチャ）

# 例を挙げる

# システム、アプリケーション運用

- BFF、API設計
  - （プラクティス）フロントエンドエンジニア側からもAPI設計を提案できる環境になっているか
  - （アンチパターン）バックエンドエンジニアだけが主導してAPIを設計しており、フロントエンドエンジニアが利用しにくい設計になっていないか

# システム、アプリケーション運用

- インフラ
  - （プラクティス）フロントエンド担当者も Web に関わるインフラの構成を把握できていて、障害対応に参画できる
  - （アンチパターン）プロダクトの特性を鑑みない画一的に用意された構成が強制されてしまう

# チーム、組織

# チーム、組織

- 専門組織の運用
  - 流れの早いフロントエンド領域に対応するためにやっておいたほうがいいこと
  - 例えば専門組織を配置し、横断的に関われるようにする、など
    - いわゆるイネーブリングチームのような運用を想定

# チーム、組織

- 分担、職務スコープの定義
  - フロントエンドエンジニアはコミュニケーションのハブになりやすい
  - 一方で隣接組織の仕事も任されやすい
    - 例) ディレクション業務をフロントエンドエンジニアが行っている
    - 例) マーケティング組織側で必要なGAの調査や設定をフロントエンドエンジニアが行っているなど
  - メンバーが本来発揮すべきバリューを阻害してしまうようなケースがないようにしたい

例はまだなし・・・
鋭意作成中！

こんな感じで鋭意作成中です

今後なんとか年内くらいに
第一弾出せるようにします

ただこの基準はあくまで
基準であって「これが達成できていないから駄目な開発」
という風に捉えてほしくないです。

本当の意味で強い組織は画一的に基準に従う組織ではなく、基準を柔軟に捉えて自分たちにカスタマイズできる組織だと思っています。

自分が見てきた開発の中で
困ってる人たちを今後少しでも助けになればと思います。

# 今後

自分の観測範囲ではどんどんリアーキテクトなどが求められているが、正直そこまで毎回必要なのか？と思ったりします。

フロントエンドのライブラリやフレームワークの更新が大変で・・・

というのをよく聞きます。

Xxx がオワコンだから、新しいyyyに乗ろう。ではなく、そもそもリアーキが必要なのかを考えられるような組織やメンバーを作っていきたいなと思っています。

そうじゃないと
フロントエンド開発そのもの
が自重で潰れるんじゃないか
と懸念しています。

色んなアプローチを取って少
しでも良くしていきたいと
思っています。

# まとめ

# まとめ

- 色んな開発組織を見てきたが、まだやりたいことができているという組織は少ない
- その原因は様々だが、ビジネスと開発のあるべき姿のギャップで困っているように見える
- こんな問題は一つのソリューションでなんとかするような問題じゃない
- ボトムアップにメンバー一人ひとりが成長する方法とトップダウンにその方向を示す方法が必要
- トップダウンにやっていく方法としてフロントエンド版のDX Criteriaを作っている
- いくつか紹介したので、頑張ってこれから続けていきます。フィードバック等があれば教えてください