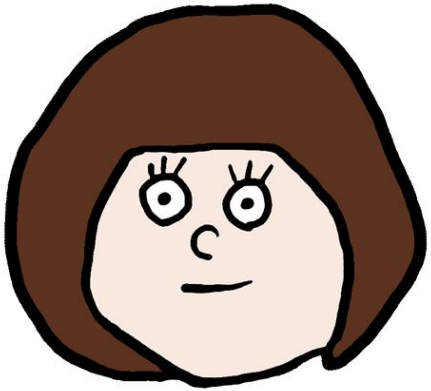
4年間でのフロントエンド リアーキテクチャの変遷と進め方

2024/12/12 Tech-Front Meetup

株式会社ナレッジワーク @yoshiko

_KNOWLEDGE WORK

よしこ @yoshiko_pg



株式会社ナレッジワークのフロントエンドエンジニア
GUIをSPAとして作るのが好き。

自社での技術スタックや設計をZennで公開しています。
https://zenn.dev/yoshiko

価値を提供する機能開発と
技術面の健全化を両立させてきた事例紹介

2020年に立ち上げたプロダクトを4年半グロースさせ続ける中で、
ほぼ自分一人の時期にそれをどう進めてきたか？をひとつの事例として共有します。
参考になる部分があれば。

- リアーキテクチャの変遷
- 技術面の健全化をどう進めたか

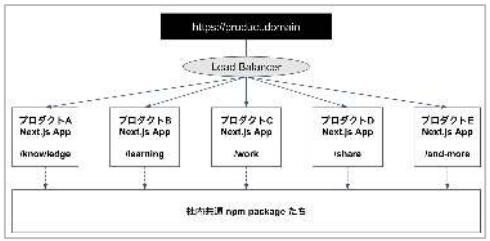
リアーキテクチャの変遷

- **モノリス最低限期**（2020〜2021）
- **モノリス拡張期**（2022〜2023）
- **アプリケーション分割期**（2024〜）

- **モノリス最低限期**（2020〜2021）

- **モノリス拡張期**（2022〜2023）

- **アプリケーション分割期**（2024〜）



中身は詳しく話す時間がないので記事などで…

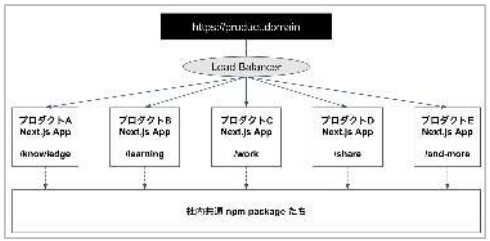
- **モノリス最低限期**（2020〜2021）

個人で

- **モノリス拡張期**（2022〜2023）

全社で

- **アプリケーション分割期**（2024〜）



- **モノリス最低限期**（2020〜2021）

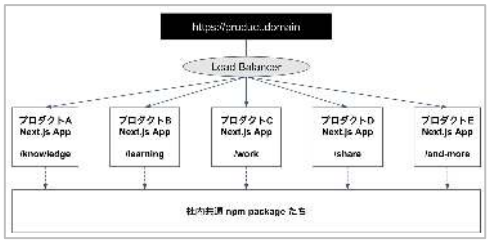
個人で

主にここの内容

- **モノリス拡張期**（2022〜2023）

全社で

- **アプリケーション分割期**（2024〜）



最低限期 → 拡張期 の技術的健全化（アプリケーションコード）

_KNOWLEDGE WORK

	最低限期（2020）	拡張期（2022）
レイヤー設計	usecaseとmodelのみ ロジックはusecaseに全部入り	usecase, repository, converter, model にレイヤー分割
APIリクエスト	fetcherにAPI Pathを直で指定 req/res型はAPIごとに手動でマッピング	API Pathを隠蔽したAPI Clientを protoファイルから自動生成
ID型	全てstring model間で取り違える可能性あり	Branded Typeでmodelごとの ID型を作成
Form	設計方針がなく、構造が複雑化 react-hook-form v5	設計方針を立て、それに沿ってrewrite react-hook-form v7 + zod

最低限期 → 拡張期 の技術的健全化（それ以外）

_KNOWLEDGE WORK

	最低限期（2020）	拡張期（2022）
core package version	React v16 Next.js v9	React v18 Next.js v13
Storybook	CSF 3.0 未リリース インタラクションテスト 未リリース	CSF 3.0 でrewrite インタラクションテスト 導入
tsconfig	"noUncheckedIndexedAccess": false "useUnknownInCatchVariables": false	@tsconfig/strictest 適用
ビルド	babel (Next.js)	swc (Next.js)
テスト	ユニットテストとVRTのみ 自前のmock機構	E2Eテストで全リソースのCRUDをカバー MSWの一部導入

技術面の健全化をどう進めたか

1. 課題の洗い出し
2. 優先度決め
3. ロードマップ作成
4. 実行

やったこと

1. 課題の洗い出し
2. 優先度決め
3. ロードマップ作成
4. 実行

プロダクトの1→10するときの思考と一緒だな…

1. 課題の洗い出し
2. 優先度決め
3. ロードマップ作成
4. 実行

意識していたこと

技術面の健全化であっても課題ドリブンで進める

ありがちな落とし穴「リソースに効果が見合わない」に陥ることを避ける

通常の機能開発でコードを書いていて　不便なこと、壊れやすいこと、非効率なことなど、
具体的に課題に感じたことを都度書き出しておく

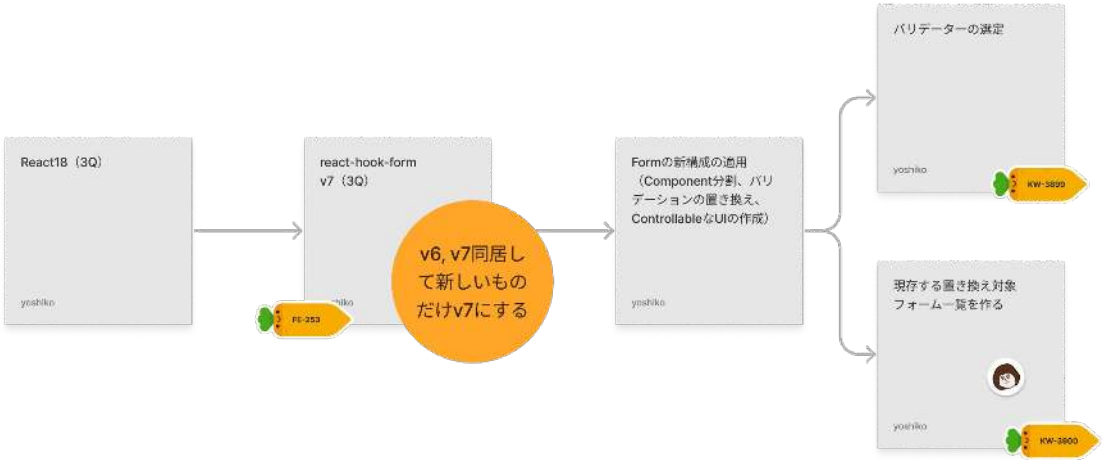
なんか思ったらとりあえずJIRAにメモ起票など

※当時はフロント設計を検討する人が自分だけだったので、とりあえず自分がわかればよかったので雑に

1. 課題の洗い出し
2. **優先度決め**
3. ロードマップ作成
4. 実行

課題一覧を見返して、課題感の大きなものや一緒にやれそうなものをピックアップする
依存関係や関連がある課題も多いので、作図ツールなどで依存も表すとわかりやすかった
大きくなりそうなものはタスクのブレイクダウンもしてみたり

例えば：
「formコードの複雑化による保守性の低下」と「formライブラリのバージョンアップ」と「Reactのバージョンアップ」に依存関係を見出した図



[実際のfigjamボードを見せます！]

※ 誰かに見せる予定のものじゃなかったので、わかりやすくまとまっているわけではないですが…

その年に着手したいことがリストアップできたら
ざっくりとカテゴライズしてみると、「方針」として自分も他人も理解しやすくなる。

社内資料「Frontend Year-end Review 2022」から実例を紹介します

重要技術課題のリストアップ

当時の社内資料から引用

① コードの書き方の一貫性

② 技術的負債の返済

③ 最新バージョンのフレームワークへの追従

④ 実装難易度の高いリッチな UXの実現

認識している課題を洗い出し、テーマとして4つの項目を定めました

この1年を通して実現したい状態

当時の社内資料から引用

	2022-01	2022-12
① コードの書き方の一貫性	同じことを実現するための方法がバラバラで新規入社者がキャッチアップしづらい	「これをしたい時はこう書く」が明確で新規入社者の立ち上がりが早くなる
② 技術的負債の返済	短期実装した複雑な機能が技術的負債となり書いたメンバーしか触れなくなっている	コードの品質に大きなムラがなく、誰でも読めば理解できるコードになっている
③ 最新バージョンのフレームワークへの追従	UXや開発生産性の向上に寄与するような使いたい新機能が出てもすぐに使えない	有益な最新機能をいつでも使うことができる
④ 実装難易度の高いリッチなUXの実現	最適なユーザー体験を提供できていない	最適なユーザー体験を提供できている

開発生産性の向上＝競争優位性(①　② ③)と 顧客便益(③　④)の実現を目指します

2022年初時点でのフロントエンド重要技術課題

当時の社内資料から引用

コードの 書き方の一貫性	- Resource Set 新アーキテクチャへの移行 - Form Componentの設計方針確立・移行 - UI Componentの拡充
技術的負債の返済	- トップページエディタ・ビューワー - バックグラウンドアップロード
最新バージョンのフレームワークへの追従	- React18の利用 - Next13の利用
実装難易度の高いリッチなUXの実現	- 入れ子構造リストのドラッグ＆ドロップでの並び替え

data source: 2022-01-05 14:04 https://www.figma.com/file/m83LTrRqaxMsxPkKLPsUTj/

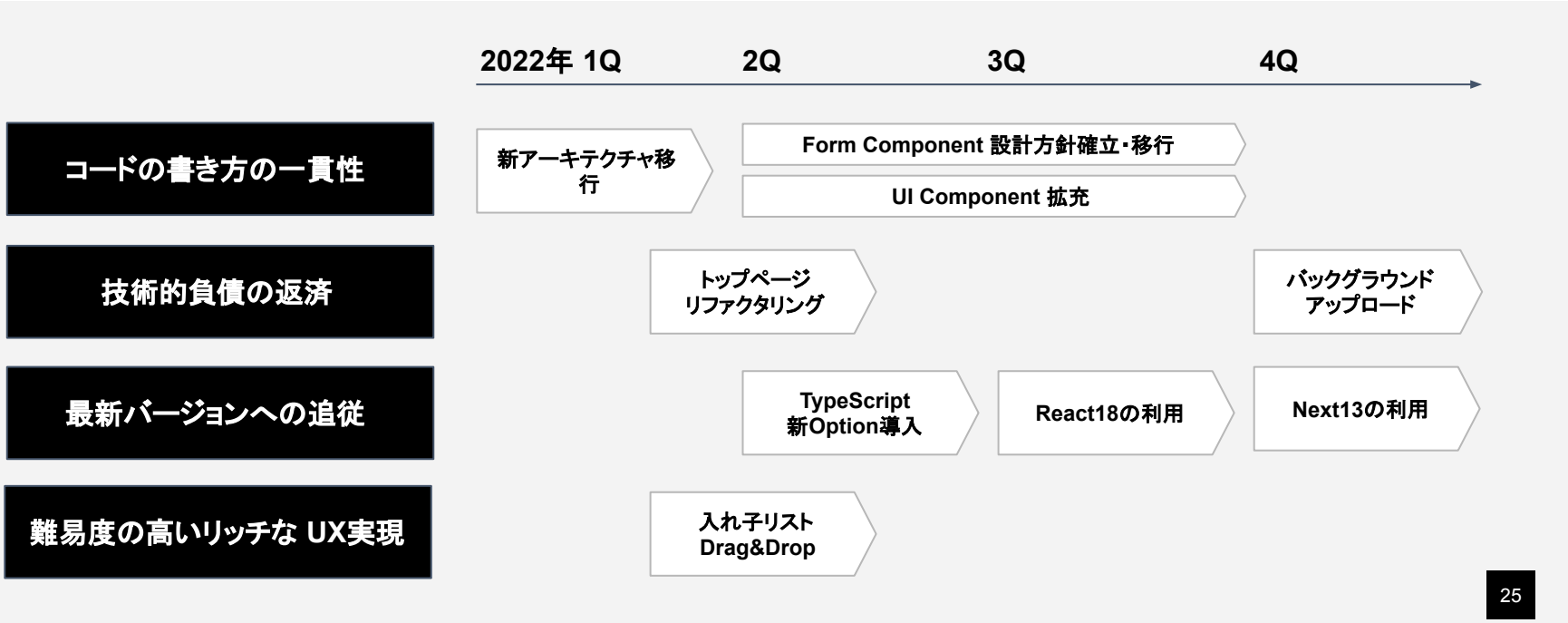
1. 課題の洗い出し
2. 優先度決め
3. **ロードマップ作成**
4. 実行

3. ロードマップ作成

_KNOWLEDGE WORK

優先度とリソース見合いで、何をいつ頃やりたいか決めていく

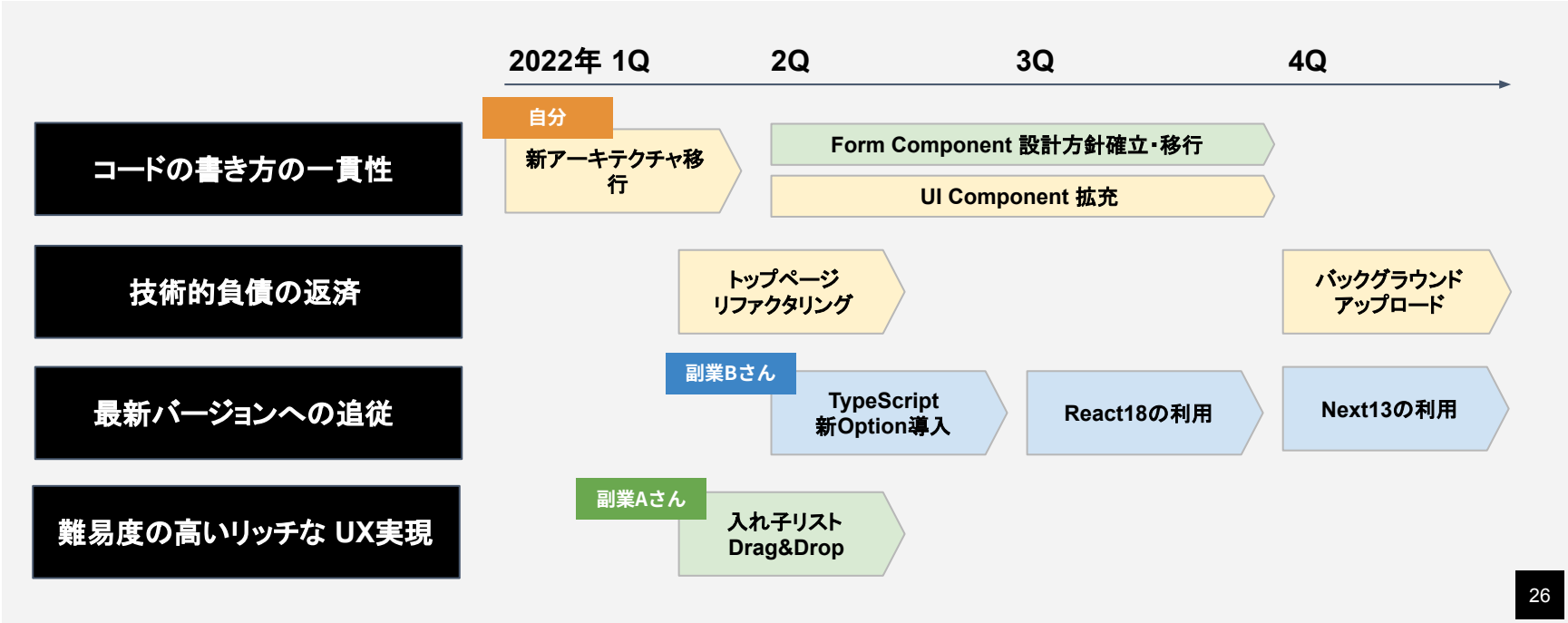
↓ は2022年の実例



3. ロードマップ作成

実現できないと意味がないので、いつ誰がどれを担当するかのイメージも持っておく

もしくは副業/業務委託採用などの



3. ロードマップ作成　～

_KNOWLEDGE WORK

当時は正社員が1-2名しかおらず、機能開発がメインタスクだったので、ロードマップの実現には人手が足りなかった。
副業募集をオープンして、面談・面接しまくって人員確保しました

副業募集には30件ぐらい応募がありました。ロードマップを元に、具体イメージのわく要件を書いてたのがよかったのかも？

月20〜40時間程度（週5~10時間程度）のイメージで募集しています！ 実稼働時間ぶんの時給制です！

■ リファクタリングが好きな方
やりたいリファクタリングたくさんあってチケット積んであります！
自由に取ってやっていってもらえたら嬉しいです！（もちろん提案も歓迎）

■ 実際に使われているプロダクトの細かいユーザビリティ改善が好きな方
やりたい改善たくさんあってチケット積んであります！
自由に取ってやっていってもらえたら嬉しいです！（もちろん提案も歓迎）

■ テストが好きな方
ユニットテストのカバレッジが全然足りてません！
Componentのテストは現状ビジュアルリグレッションテストしかありません！
E2Eはplaywrightで稼働基盤と基本的なテストはあって毎日回っていますが、カバレッジは高くないので、テストケースが増えていったら嬉しいです！

■ デザインとの協調の整備に興味がある方
デザインとフロントエンドのより強い協調を仕組みから支えたいです！
「このComponentこうしたほうが汎用的じゃない？」「ここととここのデザインこうまとめたほうがよくない？」みたいな提案や、実際に手を動かして改善していってくださる方がいたら最強です！

1. 課題の洗い出し
2. 優先度決め
3. ロードマップ作成
4. **実行**

- だいたい自分はプロダクト開発のチケットに追われていたので、自分とは別レーンで技術面の健全化タスクを動かす意識
  - そっちはスケジュールマネジメントやレビューだけ入って実働は副業の方に
  - 実働は副業の方にお願いできても、最終責任は社員（自分）が持たなきゃなので、動作確認や影響範囲の把握などはちゃんと自分がしておく。

	Read (利用List)	Read (利用Detai	Read (管理List)	Read (管理Detail)	Create	Update	Delete	その他
スペース	○	○	他テスト内で	-	○	○	○	
ナレッジ	他テスト内で	○	-	-	○	○	○	ダウンロード
ユーザー	-	-	他テスト内で	-	○	-	○	
グループ	-	-	○	○	○	○	○	メンバー追加

たとえばE2Eテストの拡充なら、実働は依頼しつつ、
こういった管理シートを作って依頼内容の可視化と進捗確認をできるようにした

- とはいえ自分がやったほうが効率のいいタスクもある
- そういうときは機能開発とそのタスクの開発箇所が重なるタイミングを見極めて、
  機能開発の文脈でリソースを確保したり
    - この機能、あのリファクタタスクやってからならもっと低工数でできそうだな…
      そしたらそのリファクタタスクも機能実装のうちでしょ！理論

むすび

- プロダクトの機能開発と技術面の健全化を並行させることで、健全なスケールをしてこれたと思っています
  - どちらかだけではなく、ANDでとるのが大事
- リードエンジニアにも課題ドリブンの思考とプロジェクトマネジメントスキルが強く活きてくると感じました

そんな健全な現場で一緒に働いてみませんか？

カジュ面フェス開催中！
カジュアル面談はこちらから！

できる喜びが巡る日々を届ける

Deliver the joy of enablement