

[ホーム](#) [タイムライン](#) [トレンド](#) [質問](#) [公式イベント](#) [公式コラム](#) [キャリア・転職](#) NEW [Orga](#)[\Qiita x Findy/エンジニアのキャリア形成を支援するコラボサイトを公開🚀](#)[■ 新規開発や新技術の検証、導入にまつわる記事を投稿しよう！](#)

@newta

【ポエム版】新規開発のときの技術選定の考え方

[開発環境](#) [ポエム](#) [アーキテクチャ](#) [品質](#) [技術選定](#)

最終更新日 2023年07月04日 投稿日 2023年07月04日

新規開発で必要なのが技術選定です。

日々システムが作られるたびに使う技術が選ばれています。

しかし、エンジニアの専門性が深く周りの職種には分からないため、技術選定が雑に行われているケースも多いと感じています。

技術選定は将来の技術負債への影響も大きいものです。そして新規開発の技術選定のタイミングは開発の中で技術負債がゼロという数少ないタイミングでもあります。

雑に決めてもOKなときはありつつ、この大事なタイミングで雑な意思決定をしてしまうことはもったいないので大事にしたいとも思います。

しかし、そのタイミングで何が必要なのかはなかなか語られていない気がしました。

と、言うことで自分のポエムを書いてみようかなと思い立ちました。

何かの文献や根拠をまとめたと言うよりは、自分の感じていることを書いている感想なので今回はポエム。

もし、機会があれば別のタイミングで知識を深掘り、他の情報も集めて整理し、言語化をして記事にするかも知れません。

その時は【ポエム版】ではなく、【保存版】とかありますか。

でもそれはなかなか大変そうなのでまたの機会に。リアクションが多ければ、可能性は高まるかも知れません(笑)

技術選定どうやるといい？という問い



20



16

...

あるあるだけど、、この理由だけで決めたらダメな例

- 何となく流行っていた
- 興味があった
- 面白そうだった

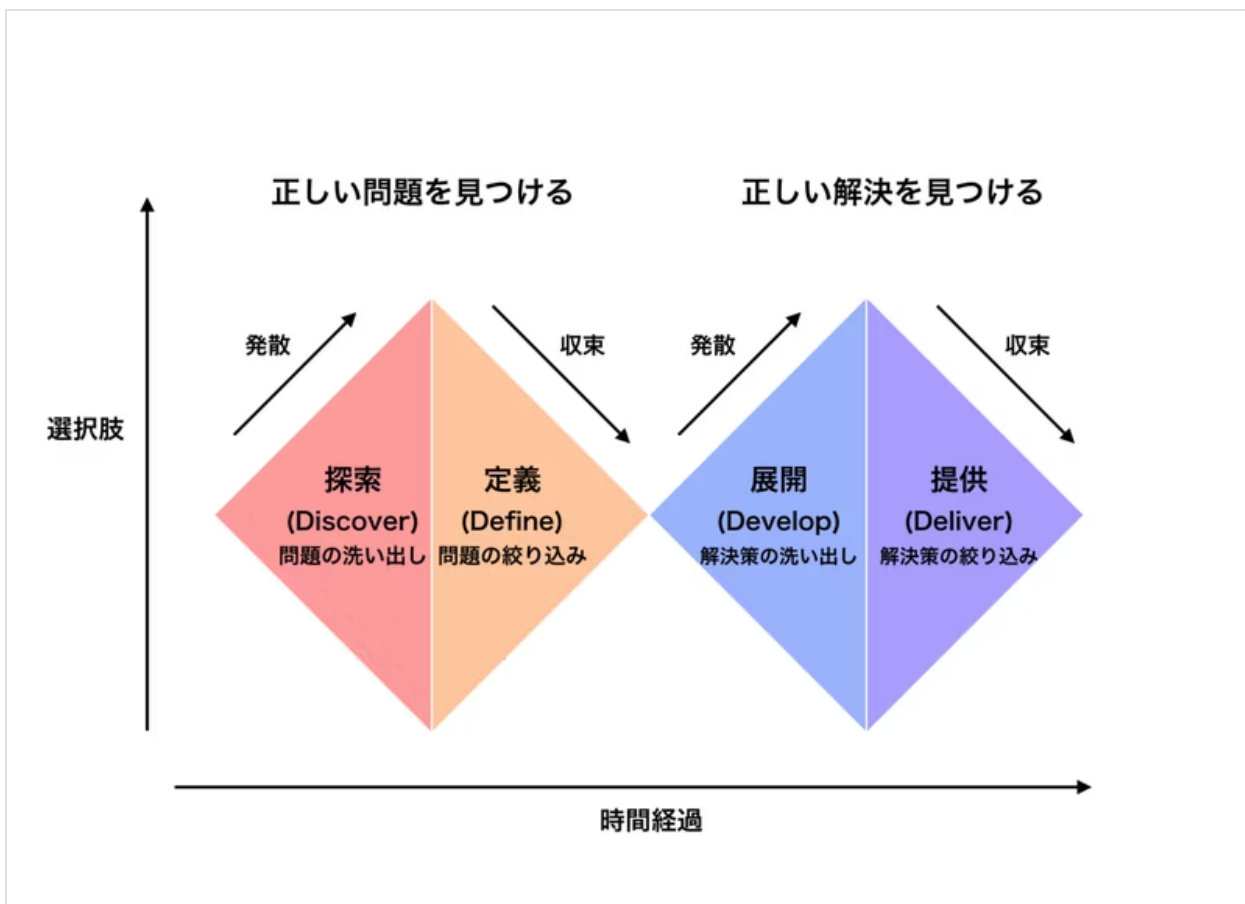
趣味のプログラミングではとても大事なモチベーションですが、仕事の場合にはもう少し深く考えるべきでしょう。

なお、これらの根拠は絶対にダメなわけではなく、最後の選択の決め手として使うのは良い方法になると感じています。

つまり、最後の選択になる前にやるべきことが何かと言うのが、この記事の話です。

技術選定という意思決定には発散と収束が重要

プロダクトデザインのダブルダイヤモンドと同じような流れが良いと考えています。



※図はプロダクトデザインのダブルダイヤモンド (UX TIMES ダブルダイヤモンド 引用)

意思決定をするときに、発散と収束が大事になります。この発散をしないままに技術を決めてしまう人がいます。



択が出来る可能性が高いですが、たまたま技術選定の場に居合わせた人は難しい可能性があります。

結果として、真のテックリードは技術選定の根拠を深く説明出来ますが、技術選定の意味を理解せずに技術を決定した場合には情報収集と検討が甘い技術選択は根拠が弱く、十分に説明が出来ないことがあります。

技術選定は将来のあり方を決定づける重要な機会であり、考慮すべきリスクを最小限に抑えながら、最大の効果を狙うタイミングの1つですが、発散が十分でなければ雑な意思決定になってしまいます。

とはいえ時間を掛けすぎても行けないから難しいですね。

とにかくまずは情報を集めることが重要です。

十分な情報がない中ではどんな意思決定でも、結果を運に頼ることになります。

どのような状態が十分であるかは状況によるため、それこそ期待するスキルによります。

十分に発散 (情報の収集し選択肢を集めること) を行うための項目

- 課題を整理する
 - どのような成果が欲しいか？
 - どのような制約があるか？
- どのような方法があるか？
 - 選択肢のフレームワークやライブラリ
 - 組み合わせ
 - メリット・デメリット
- メリット・デメリットの観点
 - メンバースキル
 - スケジュール
 - 継続性
 - 採用戦略
 - 組織戦略

本当は1つ1つについて全部語りたいくらいたくさん書きたいことがあるのですが、文章量が半端なくなってしまうので、もし【保存版】みたいな技術選定の記事を書くときがあれば個別に書きたいと思います。



- 事業
- システム
- チームメンバー
- 会社組織
- 採用

という観点と、それぞれにおける

- 過去
- 現在
- 未来

による予測ということが大事ということだけ説明しておきます。

表の例は以下です。

選んだ技術	システム	事業	メンバー	組織	採用
XXX言語	◎	○	△	X	○
XXX言語	○	◎	◎	○	△
フレームワーク	△	◎	△	△	○
フレームワーク	△	◎	△	X	◎

列の項目はここではざっくりと言った通りなので、本来であれば注目すべき観点をもう1段掘り下げたものであるほうが良いでしょう。特にシステムはざっくりすぎるかなと思います。

場合によっては個々の言語の機能を表現する必要があります。

ただ、戦略という点に置いては機能そのものよりも、機能や言語がもたらす周辺効果にこそ注目する必要があるでしょう。

次に、○やXについてです。これは、現在と未来のギャップを考えたときに良いかどうかです。

基本的に未来に向けた選択をするため、既存の会社の標準やシステムのリプレイスなどでの技術選定のときに現状維持が悪い影響を与えるときは問答無用でバツをつけるべきです。

使っている技術について尖りすぎていたり、陳腐化していたりすると、採用のスコアが悪くなります。



を選ぶべきかと思います。状況によってはシステムが△であっても採用を○である技術を選ぶ必要があるかもしれません。

純粋なエンジニアには難しい選択かもしれませんが、、、。

技術を選ぶテックリードはドライに将来のシステムのための選択を選べると良いときもあるかもしれません。

事業のスコアがXのものは選ばれる可能性がないので、メンバーの納得度を高めるための表現以外に載せる意味はないでしょう。

表にすることで観点の議論を促進したり、納得度を高められます。

集めて整理した情報から集約(意思決定)を行う

みんなで決める

技術力がある人が1人で決めるパターンもありますが、ほとんどの場合は実装するみんなで技術選定も行うほうが良いでしょう

「みんなの理解を深める」意味と1人では気が付かない「知見や知識を組み合わせるブラッシュアップする」意味、という2つの意味でみんなで決めることが良いでしょう

しかし、多数決や全会一致などの数で決めることは悪手です。数ではなくリスクや成果の大小を中心とした質の議論にしましょう

議論をし尽くしても決まらない場合の最後の選択肢として、技術の好き嫌いで選ぶタイミングであれば多数決も良いと思います

説明ドラフト版を作る

全ての情報を見せても議論はできません。表などでポイントだけを伝えたあとに、根拠の説明が必要です。

調べた大半の情報はドラフト版には現れませんが、根拠を説明する背景を持っていると説得力に厚みがあります。

議論のドキュメントは全てドラフト版であり、決定版ではありません。全てが議論によって変わる前提になります。

つまり、説明ドキュメントは議論を促進することが目的であり、主張を説明するだけにとどまらず、検討した可能性を広く説明しましょう。選ばなかった理由も重要です。



実装がないと自分の頭の中にあることは相手には伝わらず、空中戦になる可能性があります

エンジニアであればコードで話せます

逆にコードで話せない人との会議は分けましょう

技術手法の使い方や機能は大事ですが、技術選択した根拠の説明が一番大事です
理想論だけや利用技術の周辺状況だけを話しても根本的な話にはならないことがあります

手法単独での話ではなく、何を解決したいのかの話の上に手法があります

コードには意図や目的が説明が表現しにくいので、ドラフト版の説明文章とセットで効果を発揮します

実装は概念的なインターフェイスやモックメソッドのみの実装でも構いません

意思決定する

システムを一番長くメンテナンスする人の意見が大事

意思決定するのはそのシステムに最も長く付き合う可能性のある人の意見を重視する方が良いでしょう

SIの発注でシステムをリリースするところまでが担当の場合は、ほとんどの場合に顧客の言うことが正しくなります

SIであっても、リリース後も継続的に開発に係る場合は自分の意見を大事にしましょう
保守メンテするときにつらい選択を矯正される開発現場にいる人は転職しましょうw

Webの場合は事業のバランスによって技術の意思決定は大きく変わります。自分が納得できるだけの情報と決意をもって決めましょう。上司でも顧客でも、システムの面倒を見ない人の意見を聞く必要はありません。

ただし、エンジニアの自分もその決定がどのような事業的意味やどのようなユーザーの利用をもたらすのかを情報の一部として理解しましょう。技術の仕様だけでどの技術を利用するかは決まりません。

逆に自分がシステムの面倒を見ないのであれば、周りの意見を重視し、自分の希望は抑えるべきです

技術選定をした結果、システムがなぜその状態で作られているか、意思決定したかを説明可能であることが大切です。意思決定のときの別の選択肢を選ばなかった理由を説明できないと責任はとれません。



技術選定のタイミングが来る前に本当はできると良いこと

技術の引き出しを増やすこと

- 日々のキャッチアップによる候補となる技術の引き出しを増やしておくこと、いざ技術選定が必要となったときに、意思決定までの時間を短く出来ることが出来ることと、意思決定の質も高めることができます
- 限られた時間で技術的意思決定をするということは相当に難しく、日々の積み上げが効果的です

技術キーワードを知っておくこと

- 細かい部分は知らなくてもいいのでたくさんのキーワードや概要を知っておくことで、必要なときにより深く調べたり、筋が良いかどうかという最初の感覚が磨かれます

技術のアンテナを立てること

- 技術キーワードを知っておくことのためにアンテナを立てておくことが良いでしょう。基本的には受け身でもどんどん自分の知らない技術単語が流れてくれる仕組みを作ることが良いでしょう。今はサイトのRSS、メーリングリスト、ツイッター、ポッドキャストなど様々な媒体と通知の仕組みがあります。

ここでは技術選定のタイミングが来るとは書いていますが、自分から何もしないと技術選定のタイミングに出会ったことがない人もいるかも知れません。仕事の環境によってはありますが、成長にとってはあまり良くないことかなと思います。

エンジニアであれば3-5年の経験を超えてきたら、この技術選定の機会を求めると良いのではないかなーと思います。仕事場でそういう機会がないのであれば、転職を考えてもいいですし、自分の実力がという話であれば学習方法を少し変えても良いかも知れません。

自分で仮想のプロダクトをイメージして、システム構成を想像して見るだけでも良いと思います。

初めて技術選定をするときは難しいかも知れませんが、エンジニアで勉強を始めた頃のようなスキル成長を感じることができると思います！

おすすめ書籍



ここまで抽象的な話が多いかなと思ひまして、具体的に何すればいいの？的な人のために、本当は場数！！とか言いたいのですが、いきなりはもちろん無理なので、書籍を読んでもらうのが良いかなと思ひ、ピックアップしました。

技術選定がテーマであるため、これまた個別技術というよりは抽象的なシステムにおける考え方の書籍が多いですが、どれもオススメです。

◆ 進化的アーキテクチャ

技術選定は未来のための意思決定です。システムはリリースしても終わりではありません。そこからがシステムの価値の発揮です

この書籍では絶え間ない変化を想定したパターンについて書かれており、この未来を想定しながら技術を選ぶことが大事です。内容はちょっと抽象的なので、これ系の本に慣れていない方はこのあとに紹介する本を先に買ったほうが良いかも知れません。

ただ、技術選定に影響する考え方として一番抽象的に必要な思考の観点の書籍とは思ふため、気持ちが合う人には読んで欲しいです。



◆ クリーンアーキテクチャ

技術選定をする責任がある人であれば最低限知っておくと良いアーキテクチャの知識ではないかと思ひます

有名なオニオン型の絵のクリーンアーキテクチャの話は非常に参考になるアーキテクチャデザインの1つではありますが、この書籍のお得なところは、その前提として書かれている知識の深さにあります。

- プログラムのパラダイムとして構造化プログラム、オブジェクト指向プログラム、関数型プログラム
- 設計の原則であるSOILDの原則、単一責任の原則、オープンクローズドの原則、リスコフの置換原則、インターフェイス分離の原則、依存性逆転の原則
- コンポーネントの考えとして、凝集性や結合について



などなど、システム設計の土台と言えるレベルの知識をクリーンアーキテクチャの図が出でくる200ページまでの間にまとめて書かれていることです。

(ポエムだし老害的な語りを始めると、コードコンプリートという1冊でもとても分厚い書籍の上下巻で身につける話が、これだけコンパクトになった印象です 笑)

クリーンアーキテクチャの図を知っている人でも買っておいでも良い、誰でも知っていて欲しい必読書かなと思います。



◆ Googleのソフトウェアエンジニアリング

技術選定はシステムだけで決まるのではなく、様々な周辺のことを考慮しながら考える必要があります

その観点でソフトウェアエンジニアリングとは何か？というそもそもの理解が大事ではあります

この書籍はソフトウェアエンジニアリングという観点で技術、文化、プロセスをこれでもかという細かく具体的なレベルで事象の意図やエンジニア持つべき意識を書かれており、どれもが参考になるものばかりです

難点は本がとても分厚くて、読むのが大変ということです(笑)

少し価格が高いように感じられますが、その辺の3-4冊くらいのボリュームと価値はあるので、むしろお得ではという気持ちになります



◆ 良いコード悪いコード

どの技術を選ぼうがもはや関係ない観点の書籍ですが、システムは良いコードを目指して書くべきです

また、技術選択によって、どんなコードが良いコードになるかは多少の表現が変わりますが、根源としてコードで表現したい意図があることは変わりません。動作だけでなく意図を組んだ表現をする方法として具体的な表現力が得られます。

多くのフレームワークやライブラリは背後にコードの設計意図があり、その意図を理解した上でコードを実装する上でこの書籍は助けになると思います

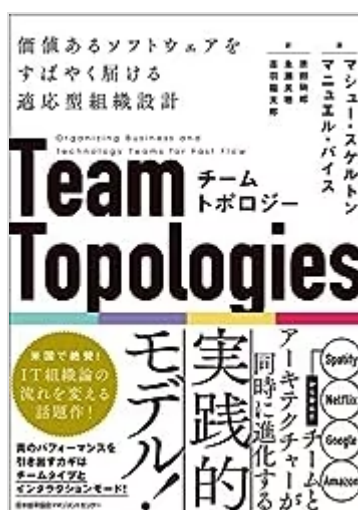


◆ チームトポロジー

単独のチームではなく複数のチームでシステムが成り立っているときに、その役割を理解するための共通言語が得られます

技術選定の影響は場合によっては自チームだけに留まらないため、システムの構造と同時に組織の構造を知れることが良いでしょう

複数チームでのシステム開発のときに自分のチームがただの作業分担なのか、継続的開発の中で役割を持ったチームなのかを考えて開発ができると良いと思います



◆ マイクロサービスアーキテクチャ



キテクチャです

具体的な技術に取り組みつつ、前項のチームトポロジー観点を持ちながら最適なサイズのシステムを作ることが出来ると技術選定の機会を多くのチームにもたらすことが出来ると思います



技術選定観点ということでアーキテクチャの本が多めですが、それぞれかなり読み応えがあるので、この記事をストックしておいてもらって1冊ずつ読んでもらえると良いかなと思います。

技術選定に参画する場合には自由にピックアップして読んでもらおうと良いと思います。が、テックリードを目指すエンジニアの人にはどの内容も全部知っていただきたい内容の本をピックアップしています。

まとめ

技術選定は非常に考慮の幅が広く、未来への洞察が必要であり、責任も重いものになっています。

でも、それは自分の技術力をフルに活かした面白い機会でもあります！

自分の技術力をフルに活用して行う機会でもあるため、エンジニアの採用面接の際には技術選定については必ず聞きます！

この技術選定に至った根拠の説明のレベルこそがその人の技術力を最も表すことの1つであると思っています。

言語やフレームワークのシステム的な特徴を説明するだけでなく、その特徴が今後においてどのような影響を与えていくのかをシステムとシステム周辺の事業、組織、文化、採用などの観点への影響も含めて意思決定するものだと知ってもらい、その上で良質な意思決定が出来るように一緒に考えられると良いと思っています。



記事について感じるがありましたら、リアクションやコメントをお待ちしています！

おまけ

別の記事で職務経歴書の書き方というものを書いていますが、この中の技術スキルを表現するための1文で

プロジェクトの中でどのような役割を果たしてきたか
どのような理解をしてその技術を使ってきたか

という表現があるのですが、これがまさに技術選定をしたときの根拠をどれだけ説明することが出来るかについて、職務経歴書の上でも表現の差として現れることがあります。

技術選定をしたときには、この技術を使いましたという結果だけでなく、どのような目的と成約があり、エンジニアとしての技術知識と思考を組み合わせた結果の意思決定を表現できることが出来るはずです。

この説明1つでエンジニアとしての質の差が分かる人は存在するということだけ理解してもらえると、もっと技術選定が上手くできるエンジニアになる目標の1つとなるかと思いました。

コメント、ご感想、リアクション、何でもお待ちしております。



0



20

16

関連記事 Recommended by



技術選定/アーキテクチャ設計で後悔しないためのガイドライン

by hirokidaichi



技術的な意思決定をする際に考えていること

by okmttdhr



fromscratchで技術選定のときに気をつけていること

by idobee



プログラミング1年目の2023年に勉強した技術・書籍の振り返り

by MoeMochiKanaoka



20

16

新生活はセキュリティの見直しから。マカフィーなら個人情報もプライバシーも安全

PR マカフィー株式会社




プログラマブルなSMS送信サービス「Karaden SMS API」企画・開発者に話を聞いた

PR NTTコム オンライン

コメント

この記事にコメントはありません。

T コメントする



プレビュー

コミュニティガイドラインに基づき、良識ある内容を心がけましょう。

テキストを入力

0B / 100MB 投稿する

How developers code is here.

© 2011-2024 Qiita Inc.



利用規約	公式イベント	Qiita マイルストーン
プライバシーポリシー	公式コラム	Qiita 人気の投稿
ガイドライン	アドベントカレンダー	Qiita（キータ）公式
デザインガイドライン	Qiita 表彰プログラム	
ご意見	API	
ヘルプ		
広告掲載		

Qiita 関連サービス

- [Qiita Team](#)
- [Qiita Jobs](#)
- [Qiita Zine](#)
- [Qiita 公式ショップ](#)

運営

- [運営会社](#)
- [採用情報](#)
- [Qiita Blog](#)

