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

発信を今後のキャリアに繋げる！Findyの「発信力レベル」活用方法とアウトプットのTipsを、C1

この記事は最終更新日から3年以上が経過しています。

@okmttdhr

# 技術的な意思決定をする際に考えていること

技術選定

最終更新日 2020年08月11日 投稿日 2020年08月10日

「人生は選択の連続である」という言葉もありますが、ソフトウェア開発も選択の連続です。

日頃から、技術やアーキテクチャの選定において、様々な意思決定やトレードオフを行う必要があります。ここでは、自分が技術的な意思決定の際に用いている考え方をまとめてみようと思います。

## 技術選定の観点

技術選定の**観点**は、例えば以下のようなものがあるでしょう。

- 主流か
- イケてるか
- 枯れているか
- 将来性はあるか
- 捨てやすいか
- セキュアか
- 情報の多さ
- 開発元、コミュニティ
- 開発生産性

31 19 •••

- GitHubのスター数
- 学習コスト
- チームの技術力
- 人材確保の難易度
- 技術広報
- 組織構造との相性
- お金(人件費や運用費など)

しかし、これらだけでは不十分です。

単に技術を選ぶだけではなく、どうしたら将来的な事業に良いインパクトを与えられるか？あるいは、事業を守ることができるか？を考える必要があります。

この記事では、様々な要素を比較検討する考え方と、技術的な意思決定についてステークホルダー(エンジニアもエンジニア以外も含む)とコミュニケーションする方法を書いてみます。

## 比較検討する手法

### プロコンテーブル

Pros, Cons, Impactをそれぞれ出し合うという、最もシンプルに合意形成ができる手法です(と思います)。

PRで議論が分かれてブレインストーミングするときなどにも気軽に使えます。ざっくばらんに意見を出し合えるところも良い点です。

| Pros | (Impact) | Cons | (Impact) |
|---|---|---|---|
| ここがいいよなあ〜 | 5 | ここがだめだよな〜 | 3 |
| ここもいいよなあ〜 | 3 | どうでもいいけどここはだめだね | 1 |
| どうでもいいけどここもいいよね | 1 | | |

## 意思決定マトリクス

課題やアイデアなど、複数の選択肢を、比較する際の手法です。

重み付けされた項目に点数を付け、その合計値を用いて、定量的・客観的に選択肢を評価します。それにより、良さそうに見えていた案が実はそうでもなかった、みたいなことをあぶり出すこともできます。

| | 緊急性 | 実現性 | 収益性 | 将来性 | コスト | 合計 |
|---|---|---|---|---|---|---|
| **重み** | x1 | x1 | x1 | x2 | x3 | |
| アイデアA | 1 | 1 | 1 | 1 | 1 | 8 |
| アイデアB | 1 | 1 | 1 | 4 | 3 | 20 |
| アイデアC | 5 | 3 | 1 | 2 | 2 | 19 |

項目に縛りはないので、例えば、フレームワークの選定などにおいて、技術的な観点をいくつか評価し、チームの合意を取るのにも有効です。アーキテクチャ選定の際には、既存のシステムとの相性や、将来的な理想像と実現ステップなども踏まえると良いでしょう。

## ICE, PIES

**ICE (Impact, Confidence, Ease)** と **PIES (Potential, Impact, Ease, Score)** は、課題やアイデアなど、複数の選択肢の優先順位を決めるフレームワークです。

`Impact` と `Ease` は共通しますが、**ICE**では `Confidence` を指標に取ります。ソフトウェア開発では不確実性がつきものなので、ICEが有効な時は多いです。例えば、どうしても失敗できない場合や、早く完了させたい場合は、 `Confidence` の重みは大きくなるでしょう。

| | Impact | Confidence | Ease | 合計 |
|---|---|---|---|---|
| **重み** | x2 | x3 | x1 | |
| アイデアA | 1 | 1 | 1 | 6 |
| アイデアB | 2 | 5 | 3 | 22 |

31 19

**PIES**では `Potential` を指標に取ります。例えば、ソフトウェアであれば、そのプロダクトの運用コスト(運用難易度)と収益性などを考慮して `Potential` のスコアを決める、などがあります。

| | Potential | Impact | Ease | Score |
|---|---|---|---|---|
| **重み** | x3 | x1 | x2 | |
| アイデアA | 1 | 1 | 1 | 6 |
| アイデアB | 1 | 5 | 3 | 14 |
| アイデアC | 5 | 3 | 1 | 20 |

## ハイ・ローマトリクス

複数の選択肢を、相対的に可視化、評価します。例えば、以下のようなマトリクスがあるかもしれません。

| | 開発コスト:高 | 開発コスト:低 |
|---|---|---|
| **将来の技術的負債:高** | Bad | Not Good |
| **将来の技術的負債:低** | Not Bad | Good |

| | CPU効率性:高 | CPU効率性:低 |
|---|---|---|
| **プログラムの簡潔性:高** | Good | Not Bad |
| **プログラムの簡潔性:低** | Not Bad | Bad |

| | システムの結合度:高 | システムの結合度:低 |
|---|---|---|
| **技術的難易度:高** | Bad | Not Bad |
| **技術的難易度:低** | Not Bad | Good |

意思決定マトリクスのように点数を出すわけではないので、相対的に比較検討するのに向いています。また、視覚的にわかりやすいので、ステークホルダーに材料を示すの

31 19

### 緊急度・重要度マトリクス

緊急度・重要度マトリクスは、問題やタスクの優先順位、本当に必要な施策なのかを判断したい時に有効です。

| | 緊急度:高 | 緊急度:低 |
|---|---|---|
| **重要度:高** | Must Do | Should Do |
| **重要度:低** | Should Do or Don't | Don't |

### ペイオフマトリクス

ペイオフマトリクスは、施策の実行是非の判断に使うことができます。

| | 実現性:高 | 実現性:低 |
|---|---|---|
| **効果:高** | Good | Not Bad |
| **効果:低** | Not Bad | Bad |

### リスクマトリクス

リスクマトリクスは、エラーや技術的負債への対応是非の判断に使うことができます。

| | 影響度:高 | 影響度:低 |
|---|---|---|
| **頻度:高** | High Risk | Moderate Risk |
| **頻度:低** | Moderate Risk | Low Risk |

## 価値を明らかにする手法

### Buy or Build

31 19

| | Pros | Cons |
|---|---|---|
| **Buy** | 即座に課題を解決できる | スケーリングが困難なことがある |
| | 初期投資が少なくて済む | 痒いところに手が届かない |
| | エンジニアを雇う必要がなくなる(こともある) | ベンダーロックイン |
| | | ランニングコストがかかる |
| **Build** | カスタマイズすることで独自の課題を解決できる | 開発コストがかかる |
| | 要件が変わっても柔軟に対応できる | 時間がかかる |
| | パフォーマンスを調整できる | 複雑度や難易度が上がる |
| | ユーザーのニーズに応えやすい | |

`Buy or Build` を判断するマトリクスは、例えば、以下のようになるでしょうか。アプリが複雑ならどのみち `Build` することになりそうですが、重要度が低ければ、コストを払ってまで作りますか？という問いは必要です。

| | **アプリの複雑度:高** | **アプリの複雑度:低** |
|---|---|---|
| **事業へのインパクト:高** | Build | Maybe Buy |
| **事業へのインパクト:低** | Maybe Buy, Possibly Build | Buy |

実際は `Buy or Build` の2択ではなく、ある程度複合したアーキテクチャになることがほとんどです。

SaaSのような「いわゆる `Buy` 」の選択肢もありますし、OSSなどを使うことは `Build` ですが、ある種のカスタマイズ性は捨てているはずです。AWSなども `Buy` ですが、よほどの規模でない限りクラウドなしの開発は考えられません。最近流行りのNoCodeも `Buy` ですね。

いずれもユースケースにフィットすればそれが1番なので、よく考えて技術選定することが大事だと思います。

31 19

### ユーザーストーリーマッピング

**ユーザーストーリーマッピングとは、ユーザーにとっての価値(ストーリー)を整理してゆく手法です。**

横軸をユーザー体験の時系列、縦軸を優先度順の体験として、付箋などに書き出してマッピングします。

- 効率よくチームの目線を合わせるユーザーストーリーマッピングのススメ

これを行うことで、価値のあるストーリーが明確になるため、どこに時間を費やすべきか判断しやすくなります。

### ゴールデンサークル

**ゴールデンサークル**は、人に情報を伝える時、Why/What/Howの3つの円が存在し、**Whyから始めることの重要性を説くフレームワークです。**

これは前述のユーザーストーリーマッピングと組み合わせると、開発するプロダクトの価値をより明確にすることができます。開発をする前にはこれらが明文化され、実際に手を動かすチーム間で共有されることが望ましいです。

例えば、具体的なHowを考えるのは開発者の仕事になりそうですが、Whyが明確になっていることで、無駄なものを開発してしまうリスクを小さくなりますし、MTGがあらぬ方向に向かってしまうことを防ぐ効果もあります。**「エンジニアの仕事はいかに開発しないかだ」という言葉もどこかで聞いたことがありますが、まさにそのための手法とも言えます。**

## 見積もりの手法

見積もりにもいくつか手法があります。

### ポイント見積もり

**基準のタスクに「ポイント」を定めて、その相対値で見積もる手法です。**スクラムではストーリーポイントと呼んだりします。

ポイントでの見積もりははフィボナッチ数列が使われることが多いですが、例えば

ベロシティを測るには数字の方がいいですが、単純に比較材料として使いたいだけなら、Tシャツサイズの方がラフに見積もることができます。

- ストーリーポイントとはどのようなもので、どのように見積もりますか | Atlassian

### SRSS法

**SRSS法 (Square Root Sum of Squares) はバッファ込みの見積もりをする手法です。**

「特に問題もなく進んだ時の平均的な見積もり」「見えうる不安を全て積んだ最悪の見積もり」のふたつを出し、それぞれの標準偏差をバッファとします(何かしらのツールでテンプレートを作っておくと良いでしょう)。これは 時間見積もり にも ポイント見積もり にも使える手法です。

ウォーターフォールのイメージはありますが、アジャイルでもおおよそのリリース時期を判断材料として使いたい時はあると思います。

以下の記事が詳しいです。

- 不安とストレスから解放される見積りとスケジュール方法 - Qiita

## 品質に関する指標

### QCD

**QCD** は **Quality, Cost, Delivery** を表しており、それぞれがトレードオフの関係にあります。例えば、 `Quality` がマスト要件なら、まずは `Quality` ファーストで考え、CostとDeliveryとのトレードオフで落とし所を探ってゆく、という考え方をします。

**QCDS (Safety), QCDF (Flexibility), QCDSM (Morale)** という派生もあります。

- Safety
  - もともとは作業環境の物理的な安全性という文脈なので、ソフトウェアでは働きやすさと言い換えてもいいかも知れません
- Flexibility
  - ソフトウェアではおなじみ、変更容易性と言い換えられるでしょう
  - 例えば、アーキテクチャの選定や技術的な難易度にどこまでコストを掛けられるかという観点になりそうです

31 19

- 士気ややる気と訳すので、チームや個人がプロジェクトに対して意識高く取り組めるかどうかという観点になりそうです
- 例えば、技術的なやりがいであったり、働きやすさや心理的安全性も含められるでしょう

ただし、ソフトウェアにおいては、 `Quality` を追うことが必ずしも他を犠牲にするとは言えません。テストをすることで開発効率は上がり、良いアーキテクチャを選ぶことで手戻りが少なくなります。

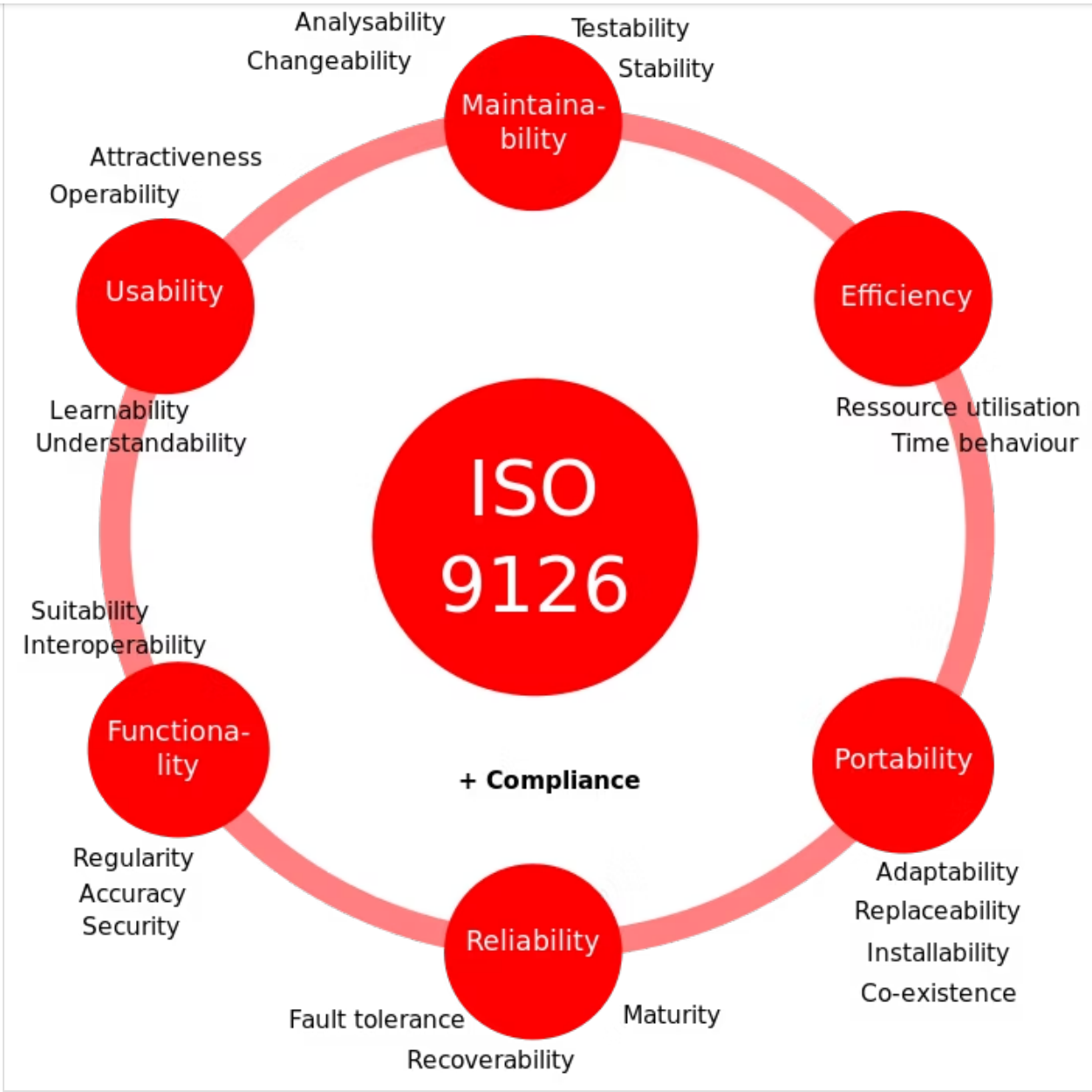
しかし、MVP (Minimum Viable Product)という言葉があるように、初期段階では過剰な機能も存在すると思います。例えば、細かなUXへのケアは品質に直結すると考えると、それらをドロップして `Delivery` を早める、などの意思決定はあると思います。

ここは状況に応じて適切な判断をステークホルダーと共有してゆく必要があるでしょう。

## Software Quality

ソフトウェアに品質評価に関してもいくつかの指標がありますが、ISOで定めているものがあったりします。

31 19




これらも評価軸やトレードオフの対象になりうるので、覚えておいて損はないでしょう。

## おわりに

ここまで、自分が技術選定の時に考えていることをまとめてみました。筋書き通りにはいかないことも多いですが、こういった指針を持っておくと日々慌てずにすむのかなと思います。参考になれば幸いです。

## 参考

31 19

- エンジニアにも役に立つ、MBAで学んだ基礎スキル 20選 - Qiita

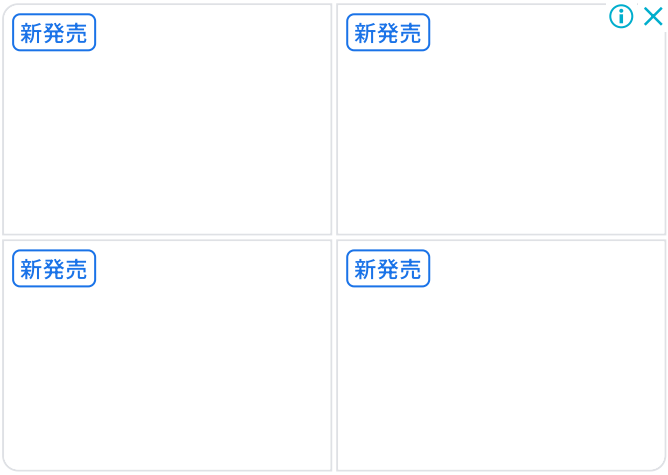






**@okmttdhr**

フォロー


新品PC・ゲーミング商品多数
ソフマップ

## 今日のトレンド記事

31 19

2024年05月18日

## Power Apps & GPT-4oを使って超高速で画像解析アプリを作る！

PowerApps AzureKeyVault PowerAutomate ChatGPT GPT-4o

♡ 37

@naoya_347 (なおや)

2024年05月18日

## useSession？なにそれおいしいの？

React Next.js useSession

♡ 25

@kaku3

2024年05月18日

## AIは仕事ではなく仕事力を奪う？

ポエム 教育 AI pm 新人プログラマ応援

♡ 15

@mkt_hanada (Makoto Hanada)

2024年05月18日

## 【AI品質・AIテストまとめ】AIシステムの品質を高めるために

テスト AI データサイエンス 品質

♡ 17

@sanjushi003

2024年05月17日

## VMware Fusion 環境 (macOS) に Windows Server 2022 仮想マシンを作成してみた

Mac Broadcom vmware VMwareFusion WindowsServer

♡ 7

トレンド一覧を見る

## 関連記事 Recommended by

31 19

by hirokidaichi

**No Estimate 見積もりしない開発手法**

by kickno

**プログラミングで一番難しいのは「見積もり」だと思う**

by yuno_miyako

**Clean Architecture 第Ⅰ部 イントロダクション**

by tak001

**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

**案件逆指名や上限なし書籍購入制度などエンジニアフレンドリーな職場環境に迫る**

PR 株式会社アスペア

この記事は以下の記事からリンクされています

**ロジカルシンキングで技術を比較検討・選定する** からリンク　7 months ago

## コメント

この記事にコメントはありません。

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。

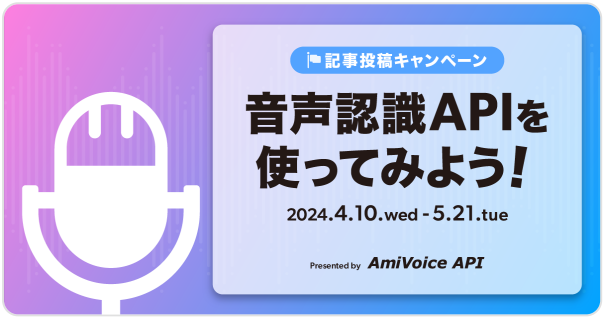
31

19

テキストを入力

0B / 100MB 投稿する

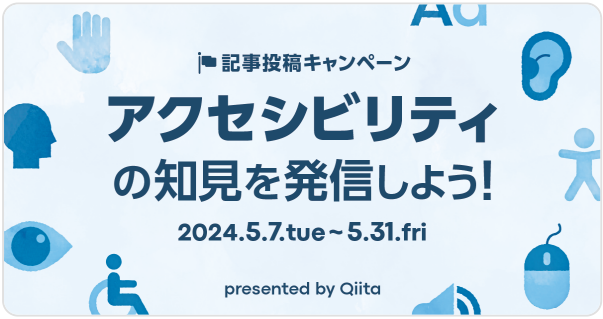
## 記事投稿キャンペーン開催中




### 音声認識APIを使ってみよう！

2024/04/10~2024/05/21

詳細を見る




### アクセシビリティの知見を発信しよう！

2024/05/07~2024/05/31

詳細を見る

すべて見る

31 19

How developers code is here.

© 2011-2024 Qiita Inc.

ガイドとヘルプ
About
利用規約
プライバシーポリシー
ガイドライン
デザインガイドライン
ご意見
ヘルプ
広告掲載

コンテンツ
リリースノート
公式イベント
公式コラム
アドベントカレンダー
Qiita 表彰プログラム
API

SNS
Qiita（キータ）公式
Qiita マイルストーン
Qiita 人気の投稿
Qiita（キータ）公式

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

31

19