

# DDDを志して3年経ったら 「DDDの皮を被ったクリーン アーキテクチャ」 になった話

---

株式会社NoSchool CTO / meijin

どんな発表？

# どんな発表？

よくある取り組みや設計ルールに対して

「無視していること」

「遵守していること」

「バランスを取っていること」

をそれぞれ事業・組織・プロダクト特性などの観点から説明します。

## つまり

- アーキテクチャ論でよく聞く「要はバランス」の具体例を徹底的に解説します！

# 自己紹介・事業紹介

# 自己紹介（1人目：名人）

## 職種・SNS

- 株式会社NoSchool CTO
- 新卒4年目でスタートアップのCTOになって5年経ちました
- Twitter(X): 名人 | マナリンクCTO
- Zenn: <https://zenn.dev/meijin>
- 好きな言語はTypeScript、好きなHTTPヘッダーはCache-Control

## 趣味

- 将棋♟️、カメラ📷、ラム酒🍷、個人開発💻、筋トレ💪、高校野球観戦⚾️



# 自己紹介（1人目：名人）②

## ZennでReact記事が人気

- Zenn記事で唯一2,000Likes超え！（登壇時点）

## 個人開発

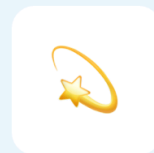
- テストメーカー（ユーザー数1万5千人以上）

## エンジニア向け教材執筆

- 「LaravelでFat Controllerをリファクタしよう」
- 本教材をきっかけに入社した社員有り

Weekly

AllTime



Reactベストプラクティスの宝庫！  
「bulletproof-react」が勉強になりすぎる件



meijin in マナリンク Tech Blog  
2021/11/08 ♡2016

テストメーカー

テストを作る

文章を穴埋めテストにして  
URLで簡単に共有・回答

文章中の大事な単語をマウスやタッチで選択！  
簡単穴埋めテスト作成ツールです。  
自学自習にも学校の授業でも使えます。

テストを作ってみる（無料）



【実践】LaravelでFat Controllerをリファクタしよう～DIコンテナ・テストコードまで～



meijin

あらかじめFat Controllerで開発されたソースコードをリファクタリングしながら、実践的なクラス設計を学べる教材です。2021年に公開後、延べ200人以上に活用いただいた教材をZennでも公開開始しました！スタートアップのCTOとして設計のトレードオフと向き合ってきた筆者が、「DDDやクリーンアーキテクチャの前にはまずここから学んでほしい」という基礎を徹底解説します

# 自己紹介（2人目：Kondo）



## 職種・SNS

- 株式会社NoSchool EM(入社4ヶ月目/まだ1年生の1学期)
- 2016年ごろにシナリオライターからエンジニアに転向
- DDDを採用しているベンチャー3社目
- Twitter(X): KondoScript | マナリンク EM
- Zenn: [https://zenn.dev/kondo\\_script](https://zenn.dev/kondo_script)
- 好きな言語はTypescriptと日本語、最近はPHPも好き

## 入社した理由

- 設計に対する音楽性の一致🎧

# 音楽性で会社を選んだ結果



→ 入社2週間でこんなやり取りをする → なんやかんやでデブサミ登壇へ至る



# 目次

- 1 私がアーキテクチャを学んだ経緯
- 2 事業・組織・プロダクトを加味した実際の設計思想を解説
- 3 4月入社 of EM から実際のところを訊いてみよう
- 4 まとめ

# 私がアーキテクチャを 学んだ経緯

CTOが「要はバランス」と言いたくなるアーキテクチャ論を、スタートアップの立ち上げを通して学んできた経緯を話しつつ、最終的に辿り着いた設計思想に話をつなげていきます。

# ざっくり経歴

2016年～2019年：新卒入社して3年間

- フロントエンド大好きな普通のWebエンジニア

2019年：CTO転職&DDDとの出会い

- 設計が分からなさすぎて一瞬で可読性ゼロのコードを量産&DDDを学びトライアンドエラー

2020年：ピボットを経て起死回生

- 事業が上手くいかなくて会社が潰れかけるが、オンライン家庭教師サービスで起死回生

2024年 設計水準を一定保ちながら事業成長を続けている

- 4月入社のEMから「DDDの皮を被ったクリーンアーキテクチャですねコレ」と評される

# 2016年～2019年：新卒入社して3年間

## フロントエンド好きのフルスタックエンジニア

- Webフロントエンドが気に入って「開眼！JavaScript」などを読む
- 一方で業務はバックエンド・インフラが多め

## 転機：社内新規事業への挑戦

- 社内新規事業提案が採択されたので、プロトタイプ開発とスーツ着て営業を1年ほどやった
- 売上も全然立たなかったし、感想としてはCEOと開発両方やるの大変すぎやろと思った

# 2019年：転職&DDDとの出会い

## スタートアップCTOに転職

- もともと教育サービスに関わりたかったので、CTOとしてスカウトを受けて素直に転職した

## 技術力なさ過ぎ問題

- フルスタックだと思っていたが、自分1人でサービス立ち上げとなると実力不足すぎた
- 知識を増やそう増やそうとしているうちに、「ダメコード」が量産された

## DDDとの出会い

- 社運が掛かった有料課金機能の実装時に、自分の実装が信用できなさすぎて、設計手法の学習とテストコードの拡充を始める。形だけの実装パターンの踏襲（戦術DDD）を始める

# 学んで速攻で実践 & 記事化

CTOになって半年。ORM依存の設計でバグを起こしまくったので**独学**で改善した知見を詰め込んだ。  
結果として300Like超をいただきモチベーションになる。

Laravelでドメイン駆動設計(DDD)を実践し、Eloquent Model依存の設計から脱却する



356



304



...



この記事は最終更新日から1年以上が経過しています。

📌 ドメイン駆動設計#1 Advent Calendar 2019 10日目

@mejileben

## Laravelでドメイン駆動設計(DDD)を実践し、Eloquent Model依存の設計から脱却する

PHP Laravel DDD ドメイン駆動設計 Eloquent

最終更新日 2021年09月07日 投稿日 2019年12月10日

この記事は[ドメイン駆動設計#1 Advent Calendar 2019](#)の 10 日目の記事です。

# 2020年：ピボットを経て起死回生

創業事業が伸びずクローズ。新たにオンライン家庭教師事業を立ち上げ

- 具体的には“来月CTOの給料出せないかも”くらい追い詰められた（2020年頭）
- コロナ禍を迎えており、オンライン家庭教師事業を立案することで資金調達して乗り切る

## 当時の私の感情

- 事業が終わった原因は何？ダメコードによるバグが足を引っ張ったかもしれないし、社運が掛かった事業で呑気にDDDにチャレンジしていたからかもしれないし、韓国から競合が参入してきたせいかもしれないし、最初のコンセプトから終わっていた可能性だってある
- 分からんからこそ「**銀の弾丸なんて無い。事業の状況、プロダクトの特性、起きている問題を直視して、そのときベストだといえる判断をしていくしかない**」と思えた

# 立ち上げた事業紹介

## オンライン家庭教師マナリンク

- オンライン家庭教師が探せるメディア（2024年現在、先生600人）
- 授業・売上・宿題等の管理ができるSaaS
- 『メディア的な機能とSaaS的な機能の両方が大事』
- Web（Next.js×Laravel）、アプリ（React Native）



結果を出すなら  
オンライン家庭教師マナリンク

500名を超えるプロのオンライン家庭教師陣が  
あなたの学力向上を徹底サポートします

こんなお悩みありませんか？

- ✓ 受験日まで時間がなくて焦っている
- ✓ テストや模試の点数が低くて悩んでいる
- ✓ 高い目標に向けてプロの力を借りたい

キャリアの長い実績のあるプロ講師だからこそ、  
お子様の弱点を明確にして短期間で結果を  
出すお手伝いをすることが出来ます。どんなお  
悩みでも、まずはご相談ください。





# カオスなオンライン家庭教師事業立ち上げの中で

## 当時のマナリンク開発の客観的な状況

- 業務委託エンジニアが離れてしまったのでエンジニアは自分含め3~4名
- 幸いにもリリース後数ヶ月で売上が立つようになったが、比例して改善が無限に発生。平行してスマホアプリの開発、旧サービスからの一部機能の移行、開発環境整備など。

## 割り切り

- 大好きなフロントエンド領域は、もうダメコード量産でOK！
- 今後事業を支えていくバックエンド（DB、インフラ含む）に注力する！
- インフラ移行期間中でコンテナベースに移行（今後数年支えるから完遂。今も生きている）
- バックエンドをDDD原理主義から適度にカスタマイズ。事業特性と重なるラインを模索

# 2020年当時のTwitterや技術記事のDDD界隈の印象

## ※個人の感想です

「DDDはこうあるべき」VS「そんなんこだわっている暇あったら機能作れ」バトル

「DDDに従っていないとまともなプロダクトは作れない的な勘違いや強迫観念」

「フロントエンドでドメイン層？何言っているの？」「スタートアップでDDDは愚策」といったコメ

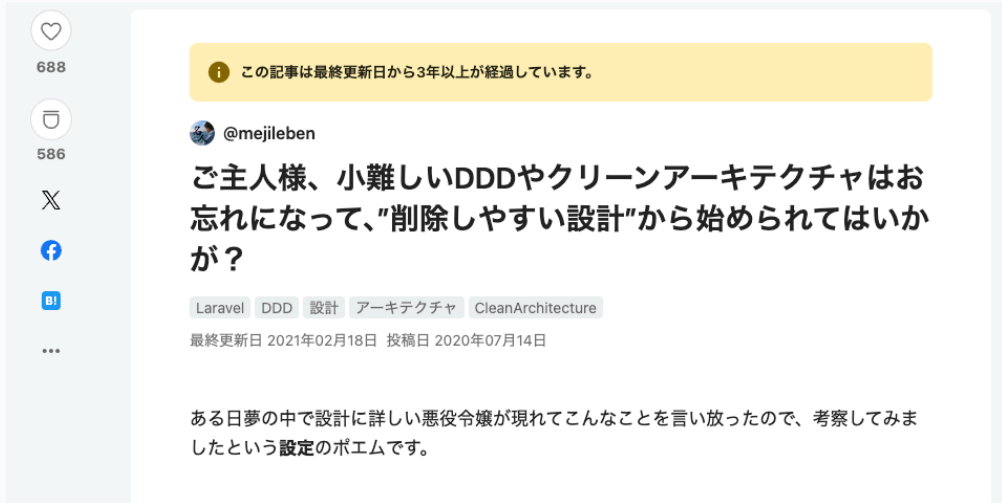
「一周回って戦術DDDでもやらないよりはマシ」「戦術DDDはアンチパターン」派の衝突

総じて、“私はこの事業でこの組織でこういう風に”「DDDをやったら上手くいきました」の””内が割愛されて広まり、誤解されている印象を受けた。発信内容を素直に受け取ればタメになることを言っているのに、前提を誤解されるケースが多そう。

# 原理主義的なネットの発信に辟易した結果...

ご主人様、小難しいDDDやクリーンアーキテクチャはお忘れになって、“削除しやすい設計”から始められてはいかが？

という煽りタイトルの記事が爆誕した



# 「削除しやすい設計」の例

“とにかくわかりやすく”単一責任原則を守れる工夫

- ControllerやUseCaseは単一のpublic methodを持つ

「DDDっぽいよね！」とか「クリーンアーキテクチャっぽいよね！」を重視しない

- あくまで「削除しやすいこと = 機能が独立して実装されていること」を重視
- 極端な話、「この処理どこに置いて良いか分からなかったので全部Entityに入れちゃいました」よりも「原理主義的なDomainServiceとは異なりますがとりあえずDomainServiceという命名で切っておきました」のほうが大事と判断

# 事業・組織・プロダクトを加味 した実際の設計思想を解説

以下考えてみましょう

なぜ弊社ではDDD・クリーンアーキテクチャを遵守しなかったのか？

なぜ「削除しやすい設計」を心がけたら一旦良いか！に至ったのか？

# 観点

どんなプロダクトを開発しているか？

ドメインの特徴は何か？

どんな規模・価値観・スキルの組織か？

技術選定は影響しているか？

どんなプロダクトを開発しているか？

# どんなプロダクトを開発しているか？

## オンライン家庭教師マナリンク

- 『メディア的な先生検索機能とSaaS的な授業管理機能の両方が大事』

## 設計思想への影響

- メディア側は色んな先生検索・表示機能がありReadのロジックが複雑
- SaaS側は授業記録や売上管理がメインで、Writeのロジックが複雑
- 立ち上げ初期からマイクロサービスとかやりたくなかったのでモノリスで通したい

**1** 機能群ごとにディレクトリを切り、それぞれDDDやシンプルな設計を使い分ける

**2** CQRSパターンの徹底・EntityをRead Modelにしない



# ①機能群ごとにディレクトリ

```
|— src/app/Domain
|   |— Calendar # 授業予定管理機能なのでDDDに寄せる
|   |— ChatRoom # ただのチャットなのでシンプルに
|   |— Learning # 宿題などの学習記録管理なのでDDDに寄せる
|   |— Notification # 通知機能なのでシンプルに
|   |— OnlineTeaching # オンライン授業機能なのでDDDに寄せる
|   |— TeacherSearch # 先生検索機能なのでシンプルに
|   ...
```

DDDの境界づけられたコンテキストやフロントエンドのPackage by Featureに近い

それぞれの機能群で互いに参照したいケースは、ひねらずに愚直に実装

- ※実践DDDなどを読んでいると、コンテキストを跨ぐときの設計手法もいろいろあるようだが、ビジネス的に境界をまたぐことが少ないこと、全エンジニアが全機能を開発するフェーズであることなどから、シンプルに参照してもスパゲティになりにくいと判断

## ②CQRSパターン・EntityをRead Modelにしない

### 事業特性

- オンライン指導の透明化をミッションにしているから、想定外のReadが後から増えがち
- **SQLけっこう速くないとSEO・CVRに影響する**から富豪的にSQLをたたけない

### 課題

- EntityにViewのための無駄なGetterが増えてただのDTOになる

### CQRSパターンの徹底

- EntityをGet時に使わない。更新のための既存データのGetには使う
- `app/Domain/Hoge/UseCase/Query` 以下に特化型のClassが大量に生える

# Queryの例

先生が見るカレンダーに表示されるデータ用のクエリ。授業予定もプライベートな予定も両方表示できる数少ない画面なので専用のQueryを用意。Query実装時は必ず唯一のpublicメソッド `handle` を持つことで多目的に使われないようにする。

```
class MySQLGetCalendarQuery implements GetCalendarQueryInterface
{
    public function __construct(
        private HogeQueryInterface $hogeQuery, private FugaQueryInterface $fugaQuery
    ) {}
    /**
     * 中略
     */
    public function handle(int $teacherId, Carbon $startAt, Carbon $endAt): array
    {
        $lessonSchedules = $this->hogeQuery->handle($teacherId, $startAt, $endAt);
        $hogeSchedules = $this->fugaQuery->handle($teacherId, $startAt, $endAt);

        return [...$lessonSchedules, ...$hogeSchedules];
    }
}
```

ドメインの特徴は何か？

# ドメインの特徴は何か？

## マナリンクは「ドメイン作りながら設計」

- オンライン家庭教師はコロナ禍以降メジャーになった新しい事業領域
- 仕様を決める際に参考になる法律や事例が無いし、ユーザー（先生・ご家庭）も“答え”を知らない
- さながら「ドメイン駆動設計」ではなく「ドメイン作りながら設計」

## 設計思想への影響

- ちゃんとビジネス側の“確度”を汲み取って設計レベルに反映する
- 「事業価値のある機能ならいずれリファクタの機会がある」というポジティブな諦め

# 「ドメイン作りながら設計」だと何が起きるか？

儲かるかわからないけどきっと価値があるから作る

- 受託やSaaSとかだと受注ベース・ニーズベースで開発するから“予算”が見えやすい
- マナリンクでは“オンライン家庭教師はこうあるべき”というリーダーシップで開発する。  
だから社内での議論を通して設計レベルを見出す

意義のある機能はどうせ改善が入る

- 初期リリースで“当たり”を引くことは少ない。利用率・内容・売上貢献などを見て即改善
- 「最初はこの設計で良いだろ。後からどうせリファクタの機会あるよ」が本当に起きる
- "機能群ごとにディレクトリ"作戦がめっちゃ刺さる

どんな規模・価値観・スキルの  
組織か？

# どんな規模・価値観・スキルの組織か？

全エンジニア3～5名（2024年現在、5名）

- 意思決定は爆速

学習意欲は高いが、手段の目的化はしないメンバー

- 「先日学んだデザインパターンをとにかく試したい」といった手段の目的化はしない(信頼)
- だからこそ、too muchに見える設計を提案されても、CTOはそれを尊重する


『ハードルを下げるのは技術選定の役目、ハードルを越えるのは各メンバーの役目』

Q. とはいえ、どうやってチームの設計レベルを底上げしているのか？



# 開発チームにどうやって設計を学ばせているか①

## 設計は“身体性”のある学び

- “暗記系”や“一度理解したらOK”の学びではなく、“体で覚える”ように学ぶ
- **自転車の乗り方**を言語化できないように、本質的には言語化&伝達できない
- 例：実装パターンだけ先に覚えてしまい、**違う、そこじゃない**ってタイミングで使われる

## “身体性”のある学びを進める方法

- トライアンドエラーの数を増やす（たくさん乗る）
- 失敗を失敗と自己理解させる（転けたら痛い）
- テックリード自ら行動で語る（大人が自転車に乗る&転けたりする）

# 開発チームにどうやって設計を学ばせているか②

## トライアンドエラーの数を増やす

- 1リリースに対して1つの設計しか検討しないより、5つ検討する方が**5倍学べる**
- 開発メンバーからの**壁打ち、相談、書籍を読んでの質問**などに徹底的に対応※出社制

## 失敗を失敗と自己理解させる

- 『機能ごとにディレクトリ』指針により、微妙な設計でもレビューを通せることがある
- できるだけ『開発者自身に2次リリース、3次リリースを主導』させ、失敗を認知させる

## テックリード自ら行動で語る

- 「要はバランス」っていう前にCTOが誰よりも設計関連の書籍を読む＆実践
- 設計失敗談や、ボツになった案を社内勉強会でメンバーに語る

技術選定は影響しているか？

# 技術選定は影響しているか？

立ち上げ初期からフロントエンドはVue or React、バックエンドはLaravel

- 設計が苦手なメンバーはしばらくフロントエンド中心にアサインするなど工夫可

## PHPの言語特性

- PHP7.4や8.0で型周り、OOP周りで相当な進化を遂げたが、一方で限界もある
- LaravelのDIコンテナが超優秀
- ほどほどにDDDやクリーンアーキテクチャを取り入れつつ、作り込むも手抜きするも自由
- 良くも悪くも**言語自体は簡単なので、より本質的な問題解決や設計に集中できる**

言語自体が難しかったり罣が多かったら、設計頑張る余裕がなくなりそうなので一長一短ですね

# 設計思想に至った背景まとめ

# 設計思想に至った背景まとめ

- 1 プロダクト特性から、**臨機応変に設計レベルを決めれることが合理的**
- 2 ドメインが未確定かつ磨くべき領域のため、**リファクタありきで設計可能**
- 3 **小規模組織**なので、世間であまり見かけない柔軟な意思決定を求めることが可能
- 4 開発メンバーが**学習意欲旺盛かつ手段の目的化はしない**
- 5 PHPやTypeScriptといった**程々に枯れた技術選定**で設計に集中しやすい
- 6 CTOが技術大好き＆プロダクト開発大好き＆事業クローズ経験者という背景

4月入社の新藤さんに  
実際のところを訊いてみよう



# トピック候補

- 1 ここまでの話を聞いてお気持ちを適当にどうぞ（適当）
- 2 この設計のぶっちゃけ課題感
- 3 この設計へのキャッチアップのやり方
- 4 実装者視点でどう見えるか
- 5 自分たちの事業や組織に合った設計レベルを見出すためのキーポイントは？



# ご静聴ありがとうございました

詳しい設計内容など聞きたい方は、この後の懇親会？で質問し  
に来てください！

X: 名人 | マナリンクCTO



X: KondoScript

