この記事は最終更新日から1年以上が経過しています。

FLINTERS Advent Calendar 2022 8日目

@Ryku in 株式会社FLINTERS

「ソフトウェアアーキテクチャの基礎」の読書メモと感想

読書 技術書 アーキテクチャ

最終更新日 2022年12月08日 投稿日 2022年12月08日

この記事はFLINTERS Advent Calendar 2022の8日目です。

はじめに

軽めの自己紹介としては、FLINTERSに入ってから2~3年くらいデータエンジニアをやってます。入社当時には現在携わっているシステムの基盤は大方構築されていて、自分の仕事としてはその運用や、細かい追加機能の開発が主で、アプリケーション全体のアーキテクチャ設計などには関わった経験はナシです。

今後のステップアップとして、設計から関わっていく機会があったときに役立つ知識をつけたいなと考えていたところに、本書「ソフトウェアアーキテクチャの基礎 ―エンジニアリングに基づく体系的アプローチ」が目についたため読んでみることにしました。そのため、アーキテクチャ素人がこの本を読んでアウトプットついでに自分なりに印象に残った部分についてまとめた程度のものになることをご了承ください。

> 私たちは、ソフトウェアアーキテクチャを、システムの構造、システムがサポートしなければならないアーキテクチャ特性、アーキテクチャ決定（「イリティ（-illity）」）、そして設計指針の組み合わせで構成されるものだと考えている。

上記のように、4つの属性で構成されるもの、と定義されている。

システムの構造

マイクロサービス、レイヤード、マイクロカーネルなど、基本的なアーキテクチャスタイルのこと。自分的にはアーキテクチャといえばこのイメージだったが、これを決めただけでは厳密にはアーキテクチャを定めたことにはならないとのこと。

例にあげた構造（マイクロサービス、レイヤード）のざっくり解説。

- マイクロサービスアーキテクチャ
  - サービスを構成している要素を「マイクロサービス」と呼ばれる小さな独立した機能に分割して実装する手法のこと
- レイヤードアーキテクチャ
  - 技術ごとにレイヤーを分けた設計のこと
  - Webアプリケーション設計の例で言うと、UI層、アプリケーション層、ドメイン層、インフラ層などに分かれる

アーキテクチャ特性

ソフトウェアアーキテクチャを定義する別の側面。可用性、信頼性、テスト容易性、スケーラビリティ、など。
システムの機能に直接関係するものではないが、システムが適切に機能するためにはこれらの特性が必要になる。

ソフトウェアアーキテクチャの法則

> ソフトウェアアーキテクチャはトレードオフがすべてだ。 -- ソフトウェアアーキテクチャの第一法則

> アーキテクトが、トレードオフではない何かを見出したと考えているなら、まだトレードオフを特定していないだけの可能性が高い。 -- 必然的帰結その一

開発時にはもちろん最善のアーキテクチャで開発をしたいと望みがちだが、本書で何回も強調されているように最善というものは存在せずに、実際にはトレードオフがあるだけ。

テスト容易性、モジュール性などが優れた設計にして、うまく構築できたと思っていても、テキトーに作るよりも工数はかかる訳でまずはコストというトレードオフが発生しているし、あとは仮にそれによってコードが複雑なものになった場合、メンバーの入れ替え時の引き継ぎの難しさなどのトレードオフも出てくるのかなと思った。

アーキテクトと開発者の違い

アーキテクトと開発者の違いについて語られている。アーキテクトはアーキテクチャの観点でものごとを見る必要があり、以下のような技術的な幅というものが重要になってくるとのこと。

技術的な幅の重要性

開発者は専門性が命。狭く（広くても良い）深くの知識が求められるが、アーキテクトは技術の幅広さが求められる。つまり広く浅くが良いとされる。
アーキテクトになったばかりでやりがちなミスの代表例が、自分が得意な技術を掘り下げる癖が抜けきれず、十分な広さを得る時間が無くなってしまうこと。

プロジェクトの成功のために

アーキテクトは開発者に自らの決定を遵守してもらうのにてとも苦労する。アーキテクチャを機能させるためには、アーキテクトと開発チームの密なコラボレーションが不可欠となる。

自分的にもこれは苦労するだろうなと容易に想像できた。ちょっとしたコーディング規約を作るのでさえ、決定とそれをチームへ浸透させる難しさというのは経験しているので、もっと大きい枠組みであるアーキテクチャを機能させるとなれば相当だろうなと思った。

コンポーネントベース思考

モジュールが物理的にパッケージ化されたものをコンポーネントと定義されている。

> コンポーネントは、モジュール化されたアーキテクチャの基本構成要素であり、アーキテクトが考慮すべき重要な要素だ。実際、アーキテクトが決定しなければならないことの１つに、アーキテクチャ内の最上位コンポーネントをどう分割するかということがある。

様々なコンポーネントが集まり一つのプロダクトができているし、そのコンポーネントも別の視点から見るとさらに細かいコンポーネントとして分割することもできる。ここで重要になるのが、どのようにコンポーネントに分割していくか、ということ。一番最初の分割、つまり最上位分割の性質の違いが、アーキテクチャの性質を決定付ける。

また、プロダクトというのは、様々なコンポーネントベースで作られており、そのコンポーネントごとにアーキテクチャが存在する。各アーキテクチャスタイル紹介の章で「マイクロサービスアーキテクチャの中の一部のサービスがレイヤードアーキテクチャで構成されることもある」というような説明が出てくるが、このコンポーネントベース思考というのが前提にあることを考えると理解しやすかった。

例えばMLOpsシステムを構築しているのであれば、MLのためのデータを格納するコンポーネント群、MLのコンポーネント群、MLの結果を特定のサービスに応用するコンポーネント群などに分割することができる。そして、それぞれが別々のアーキテクチャを持ち得るということかなと。

モノリシックアーキテクチャと分散アーキテクチャ

> アーキテクチャスタイルは、モノリシック（すべてのコードが単一のデプロイメントユニットで構成されている）と分散型（リモートアクセスプロトコルを介して接続された複数のデプロイメントユニットで構成されている）の2種類に分類できる。

本書では以下のモノリシックアーキテクチャと分散型アーキテクチャが紹介されている。

    - レイヤードアーキテクチャ
    - パイプラインアーキテクチャ
    - マイクロカーネルアーキテクチャ
- 分散型
    - サービスベースアーキテクチャ
    - イベント駆動アーキテクチャ
    - スペースベースアーキテクチャ
    - サービス指向アーキテクチャ
    - マイクロサービスアーキテクチャ

一般的に、分散アーキテクチャは、パフォーマンス、スケーラビリティ、可用性の点でモノリシックアーキテクチャよりも強力であるとされている。しかし、分散アーキテクチャはネットワークによるサービス間の通信を前提としたアーキテクチャであるため、ネットワークの信頼性への依存が大きい、モノリシックよりも構造が比較的複雑になるなどのトレードオフがある。

各アーキテクチャスタイル解説

これ以降の章で、先ほどリストアップしたモノリシックアーキテクチャと分散型アーキテクチャすべてについて、一般的なトポロジー*、特徴、メリットデメリットに至るまで詳細に記述されている。

※ トポロジー（ネットワークの接続形態を点と線でモデル化したもの）
参考：https://wa3.i-3-i.info/word12191.html

本書の中核部分でありページ数も多く割かれている。前述までの基礎知識を押さえておくことによってこのスタイル解説が理解しやすくなった。
が、実際自分はレイヤードアーキテクチャ、サービスアーキテクチャを聞いたことがある程度の知識しかなかったため、この章以降を深く理解できる段階ではないかなと感じた。ただ一応すべてのアーキテクチャスタイルについて目を通したので、今後設計に関わる際や、アーキテクトのいるチームで開発者として開発する際にアーキテクトの決定を理解するための下地などにもなれば良いなと思っている。

まとめ

- トレードオフの考え方は勉強になった。
- 実践形式で要件からアーキテクチャ特性やコンポーネントを発見していく方法（アーキテクチャ・カタと呼ばれる）も紹介されており、具体的にどう決めていくのかのイメージをつけることができた。
- いきなりマイクロサービスアーキテクチャなどの概念から説明するのではなく基礎的なところから説明があるため、理解の手助けになった。

誰が読むと良い？

基本的にはテックリードやアーキテクトの経験が一度あったりする方だと、読んで深く共感できる部分、深く理解できる部分はさらに増えてくるのかなと思うので特にオススメしたい。

自分のような1開発メンバーとしての開発経験が数年のみの人には少しレベルの高いものなるかなと思ったが、前述したようにアーキテクトのいるチームで開発者とし開発する際にアーキテクトの決定を理解するための下地にするという考え方もできる。
また、純粋なアーキテクトを持たないチームも実際は結構あるし、そういったチームでは設計思想の入り乱れたスパゲッティコードだらけのような状態を避けるためにメンバーのアーキテクチャに対する各々の理解が重要になるということもあり、アーキテクトに限らずエンジニアであれば持っておいて良い知識であるかなと思った。

次回9日目は@saaaaraさんのオフライン勉強会に関する記事です、お楽しみに。

@Ryku

本職はデータ運用ですが、趣味でWeb開発系の勉強したりしてます。 発言は個人の見解であって、所属組織を代表するものではありません。

フォロー

株式会社FLINTERS

技術を磨き続けることでネットサービスを軸とした新たな体験を世界に提供します。

https://www.flinters.co.jp/

フォロー

📈 今日のトレンド記事

@DEmodoriGatsuO (Gatsuo De'modori)

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！

4

0

♡ 34

@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

マイクロサービスの粒度
by soichiro0311

♡ 4　0

[翻訳] Shopifyにおけるモジュラモノリスへの移行

by tkyowa

エッセンシャル ソフトウェア開発

by reoring

レイヤードアーキテクチャについて

by dich1

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

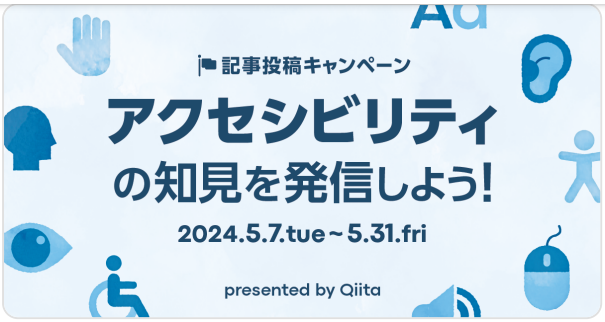
コミュニティガイドラインに基づき、良識ある内容を心がけましょう。

テキストを入力

0B / 100MB 投稿する

記事投稿キャンペーン開催中

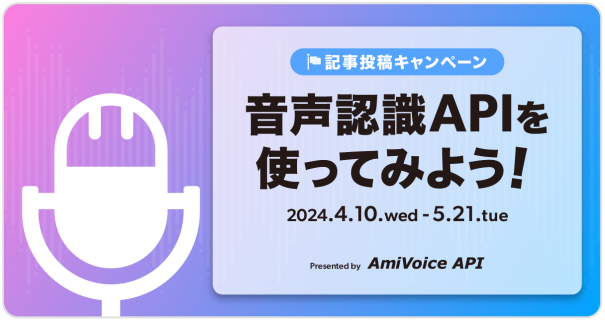
4 0



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



音声認識APIを使ってみよう！

2024/04/10~2024/05/21

詳細を見る

すべて見る

How developers code is here.

ガイドとヘルプ
About

コンテンツ
リリースノート

SNS
Qiita（キータ）公式

4 0