お題は不問！Qiita Engineer Festa 2023で記事投稿！

@Kudo_panda (せやかて 駆動)

戦術的フォーク

アーキテクチャ　#戦術的フォーク　#アーキテクチャ分解

最終更新日 2023年11月04日　投稿日 2023年07月03日

背景

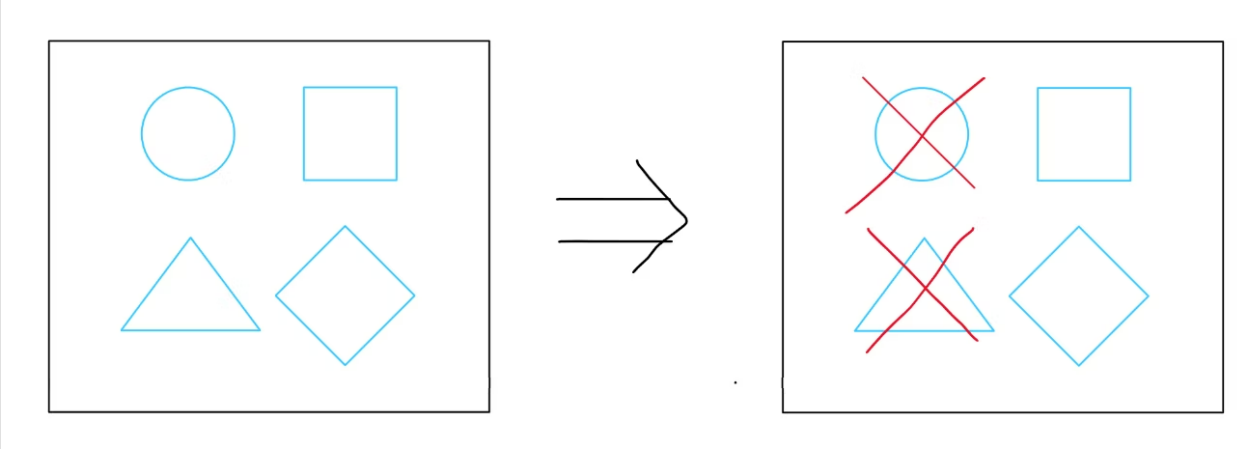
ソフトウェアアーキテクチャハードパーツという書籍の中で、アーキテクチャの分解の方法には
大きく分けて以下の方法があると書かれていた。
・戦術的フォーク
・コンポーネントベース分解

今回はこの中で前者の戦術的フォークについて記載する。

概要

それではこの戦術的フォークってどんなものなんだろう？
手っ取り早く要約すると、分業制で不要な部分を削除して必要な部分だけを抽出する手法だ。

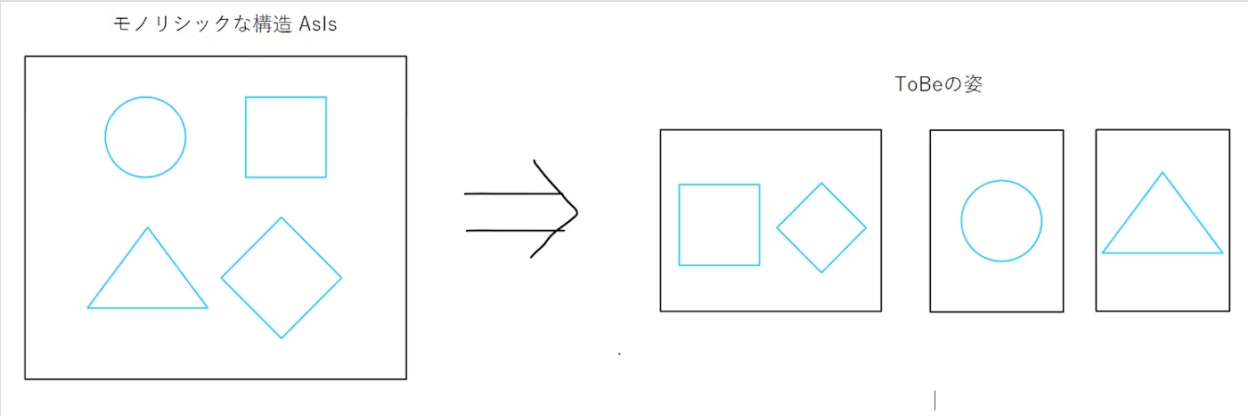
(下図を参照)



これだけ聞くと「なんだ〜〜簡単じゃないか」って錯覚するけども、意外とそうもいかない。
以下ではその注意点を詳細情報を交えながら記載していく。

AsIsとToBeの姿

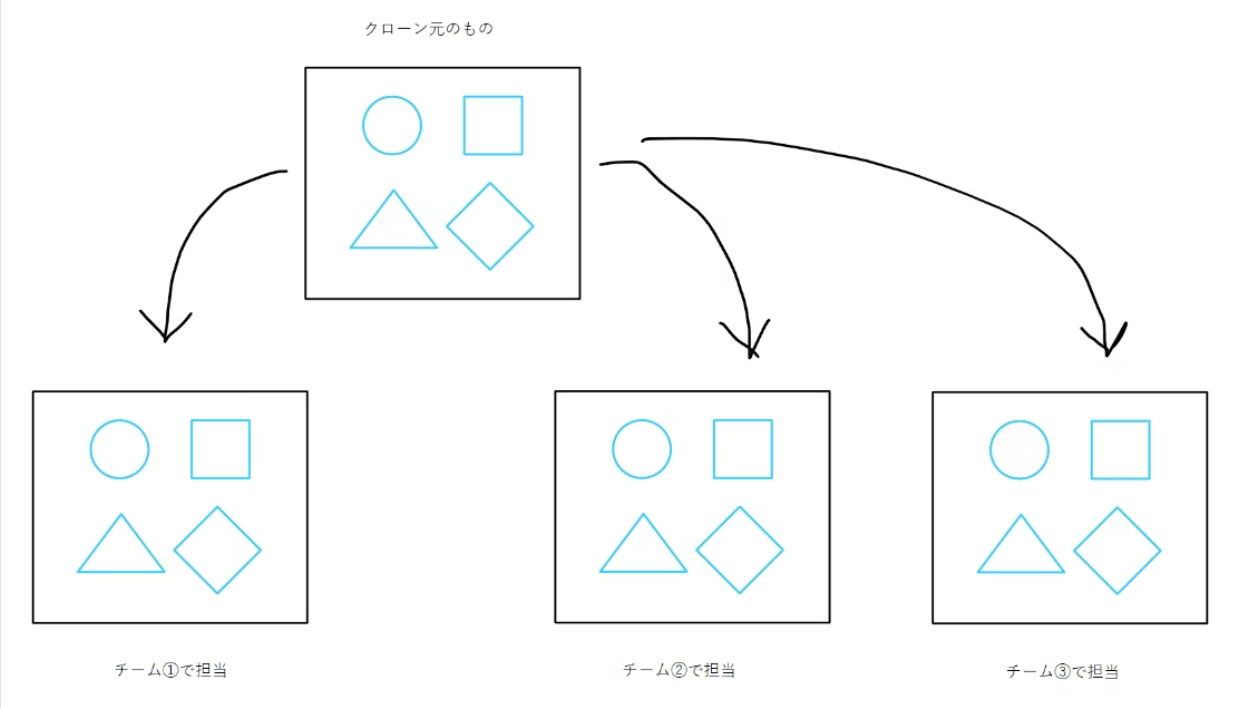
元の現状のアーキテクチャの姿が図の左側だとする。
そしてなりたいアーキテクチャの姿は右側の方だとする。
通常のリファクタリングでは、このなりたい姿のToBeからトップダウン思考で、
どのように段階プロセスを辿ると、低リスクでToBeに近づいていけるかというように考える。
しかしこの戦術的フォークでは、名前の通りそのようなトップダウン思考はなく、
ToBeの姿が分からない状態で、ボトムアップ思考でひたすらに不要なものを取り除いていく考え方である。



詳細手順

最初にクローンを用意

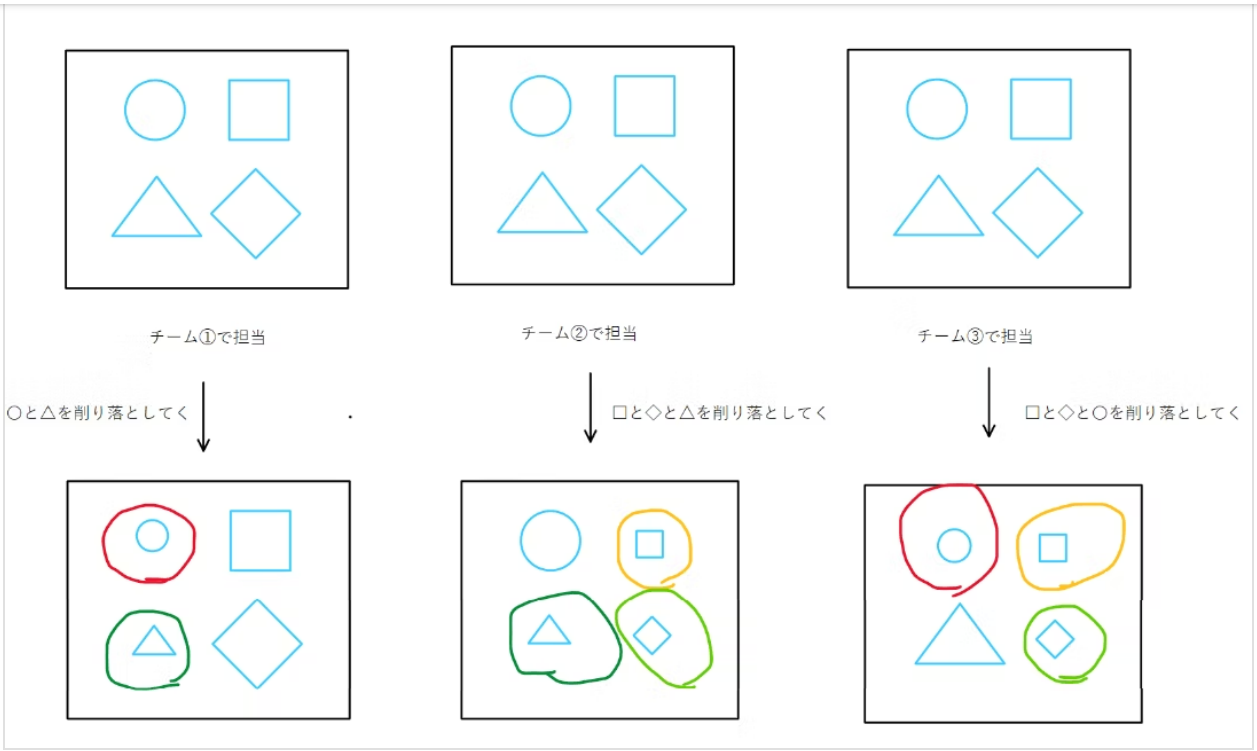
まずはいきなり本番用のものを削除するんではなく、事前に各チームに対してアプリのクローンを用意する。



小さく削ってく

仮に一気に不要と思われるものを削除した場合、後に
「ちょっと待って💦　あの削除した部分必要だったのに！！」と後になってから必要となるリスクに対応できない。
このリスクを最小限に抑えるために、削除したことによる他のパーツへの影響が出ない部分であり、
かつ今後もそのビジネスにおいて必要と思われる可能性が低いと考えられる部分から小さく小さく削ぎ落してしていく。

そのため、小さく削ぎ落してはその削ぎ落した部分が、そのアプリを使ってるビジネスで必要とされていないかのチェックをマーケティング部門の人とかと常時監視する運用体制が必要だと思われる。



注意点

ここで気を付けたいのは削り落としていく際の重複作業だ。
図において同じ色の印をつけた部分が重複作業部分になり得る部分なので、何も考えずに各チームが余計な部分の削ぎ落し作業を開始してしまうと、無駄な作業コストが発生する。
チーム①との重複作業が発生することで無駄な作業コストが発生するリスクがあるということだ。
さらに後述の短所の項目で触れるが、この際には同じ概念(コンポーネントやクラス)に対して異なる名前を付けてしまうリスクも発生する。
そのため、チーム①とチーム②でともに連携なく分業で△を削っていくという作業はやめた方がいい。

よってこの重複作業リスクと、命名不整合リスクを発生させないために、
たとえば密に連携しながら重複部分の作業を進めるといいと感じる。

ただし、これでは分担したことによる密な連携という、別チーム同士の密結合が起きてしまってる。
本来の理想は、チームを分けたのなら、別チーム間との連携は疎結合であるべきだ。

長所

様々な要素同士が密に結合しあっていたりして、まだアーキテクチャが洗練されていない状況下では、
重要なコアとなる部分などを抽出(DDD本では蒸留と表現されていた)するのは、非常に困難を極めるのは容易に想像がつくが、
その一方で不要な部分を浮き彫りにするのは、コンパイルや動作検証によってすぐに判明しやすい。
つまり、コンテキスト境界がどこかな〜〜？なんてチーム全体で議論し合うなんてことせずとも、そのアプリにおいて必要な部分だけをあぶりだすことができるわけだ。
よって、迅速にチームごとに分担してリアーキテクティングするには学習コストも比較的少なく可能と思われる。
この継続的な不要なものを削除しながら、ちょっとリファクタリング活動してはリリース。
また不要なとこ削除しながらちょっとリファクタリング活動ってのを繰り返していけば、
まだ設計能力がそこまで高くないチームでも継続的に少しずつ設計能力を上げていけるだろう。
(もちろんそういった学習期間というものが、プロジェクトの機関に組み込まれていればの話だが)

短所

以下のようなデメリットがある。
何より分業を前提としているからか、リスク回避のためにチーム間の密な連携を必要とし、
コミュニケーションコストがかかることが個人的には非常にうざったい。

①品質が上がるわけではない

単に不要なものを削除するだけでは、使われていないコードが無くなっただけであり品質が良くなったわけではない。
チーム①の人が〇の部分で品質の宜しくない部分をリファクタリングしてくれるとかなら品質は向上するだろうが、
その際にも他のチームの人が同じ〇の部分をリファクタリングするっていう重複作業を

②命名不整合によるDRY原則違反のリスク

上記の詳細手順のようにチームを分担して行うことによって、共通概念のものに対する命名がチームごとに違った命名がされるリスクが伴う。
これは①で述べたリファクタリング活動にも起因するが、せっかく自分たちが不要なものを削除しながらも、
品質を上げるために名前の再設計をするとかしていたのに(利用者から図書館利用者へリネームしたとか)、
違うチームが同じところを違う名前で設計してしまい(利用者からユーザーへ変えたとか)、
その状態で何日か経過してしまったら、もうそのチームの人もどんな意図をもってして名前を再設計したのかなんて覚えていないでしょう。
そうなってしまったら、「あれ？このユーザーてのと図書館利用者てのは全く同じ概念なの？？」なんてことになりかねないことは想像がつく。
こんなわかりやすい例なら気づけることもあるだろうが、実際の現場ではこの概念の重複に気がつかれることなく、
そのまま後になって重大な問題化するまで放置されてしまうこともあるだろう。
このリスクを回避するために、分担したチーム同士で
「あなたここ今私がリファクタリング活動してるから、そっちで勝手にいじらないでね？」
なんて密な連携が必要とされてしまう。

個人の見解

ビジネスドメインの理解が不十分な状況下で、ただ単に不要なコードを削除するだけのような
戦術的フォークパターンは、中長期的な視点で見た場合負債の回収に向いているとは思えないと感じた。
初期コストを抑えつつもメンバーの設計力を底上げしたい時などにおいては有効と感じる。
しかしながら分担したチーム間の密な連携といった、アーキテクチャ愛好家の目線では
「なんで作業効率あげるために分担したのに、そこに密結合起きてんのよ！！(#゜Д゜)」て強く感じてしまう。

システムのアーキテクチャ、ビジネスのアーキテクチャ、チームのアーキテクチャ、すべてがそれぞれ対応しており

OKといった状態が望ましい、
と非常〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜に思う。

0

@Kudo_panda (せやかて 駆動)

エンジニアとしての経験年数は3年目、アーキテクトは2年生になります。 システムオブシステムズのビジネスアーキテクトなどをしつつ、たまにイベントで登壇もしてます。 ソフトウェア領域の設計原則や思想をビジネスサイドにも適用しつつ、チームトポロジーの考え方も取り入れつつ、変化に強いビジネス構造を実現することを心がけています。

フォロー

3 0

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！

PowerApps AzureKeyVault PowerAutomate ChatGPT GPT-4o

♡ 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

3 0

@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

リファクタリング自爆奥義集
by MinoDriven

ドメイン駆動設計の比類なきパワーでRailsレガシーコードなど大爆殺したるわあああ！！！
by MinoDriven

iOSアプリアーキテクチャ比較検討(Cocoa MVC,MVVM,MVP,CleanArchtect...
by repepe2en

「設計なんて不要でしょ」について

♡ 3　0

この記事は以下の記事からリンクされています

 **コンポーネントベース分割** からリンク　2 months ago

 **戦術的フォーク、コンポーネントベース分割とECRS原則の関係性** からリンク
2 months ago

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

3　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

3 0

3 0