Engineer

コードベース

チームが一つのまとまりとして編集、レビュー、出荷するソースと関連ファイルの全体

どのようなものか

コードベースは、チームが一つのまとまりとして編集、レビュー、出荷するソース全体です。英語ではcode baseと二語で書くこともあります。エンジニアが「コードベース」と言うときは、アプリケーションのあらゆる変更が集まる場所を指します。コード本体、テスト、ビルドや検査のスクリプト、データベースの変更、各部分の関係を説明するドキュメントが含まれます。

ウェブサービスを構築するための方法論であるTwelve-Factor Appは、実務に最も役立つ定義を示しています。境界を実際に検証できるからです。Subversionのような集中型バージョン管理システムでは、一つのリポジトリが一つのコードベースです。Gitのような分散型システムでは、ルートコミットを共有するリポジトリの集合を指します。ルートコミットとは、履歴の最初にある、親を持たないコミットです。同じ最初のコミットから発展した二つのリポジトリは、どれほど違いが大きくなっても一つのコードベースです。別々に始まった二つは、ほぼ同じファイルを含んでいても、二つのコードベースです。

同じ文書は、この定義にもう一つの規則を組み合わせています。一つのコードベースから、複数のデプロイを行うという規則です。各デプロイは、アプリケーションの稼働する一つのインスタンスを指し、通常は複数が同時に存在します。本番が一つ、ステージングがもう一つで、各開発者のローカルコピーも含みます。すべて同じコードベースを実行しますが、コミットは異なることがよくあります。開発者の手元にはステージングにない変更があり、ステージングには本番にない変更があります。

中に何を含めるかは、この定義から決まります。ソースと、それを動くプログラムにするために必要なものはすべて含めます。各依存関係の名前とバージョンを示すマニフェストも、その一つです。依存先そのものは通常は外に置き、パッケ

ージマネージャーで取得します。デプロイごとに変わる値も外に置きます。特に認証情報はそうです。一度コミットしたパスワードは、その後、履歴のすべてのコピーに付いて回ります。

一つのコードベースに複数のプログラムを含めることもできます。モノレポはフロントエンド、API、データベースのコードを一つのリポジトリに並べ、一つのコミットで三つすべてを変更できるようにします。ルートコミットによる規則では、これも一つのコードベースです。逆に、別々の履歴を持つリポジトリにアプリケーションを分けると、各リポジトリが独立したコードベースになります。Twelve-Factorは、その帰結を明言しています。コードベースが複数なら、それは分散システムであり、各構成要素がそれぞれ一つのアプリケーションです。

概念を判断に生かす

「コードベースとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 まず全体像を設計し、各部分のつながりと費用を、コードを書き始める前に明らかにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 14部門を支える共通インフラ計画 — 14 共通インフラ計画の対象となった研究部門

14部門を支える共通インフラ計画 お問い合わせ →

なぜ重要なのか

境界は、何を一緒に変更できるかを決めます。一つのコードベース内なら、APIとそれを呼ぶ画面の変更を、ひとまとまりとして書き、レビューし、マージできます。二つのコードベースにまたがると、同じ変更に二つのプルリクエストと二回のリリースが必要になり、一方だけが先に進んでいる期間が生まれます。どちらも運用できますが、その負担は境界の引き方で決まります。

コードベースは、基準を適用する単位でもあります。テストスイート、リンター、フォーマッター、コミットメッセージの規則、タスクランナーは、いずれも「コードベース」を対象範囲とし、その外にあるものはどれにも検査されません。

初期の判断は、コードベースでは特に早く定着します。判断の後に書かれるすべての行が、その判断を前提とします。そのため、最初の数週間で決めた層分けの規則や品質基準が、その後に続くすべての形を決めます。後から覆すには、その前提に沿って増えたコードを書き直すことになります。

人にとって、コードベースは新しく加わった人が受け取るものです。ビルド、テスト、実行に必要なものがすべて入っていれば、一度クローンし、一つのコマンドを実行すれば始められます。必要な知識の一部がコードにあり、一部が誰かの記憶にあるなら、最初の一週間は、何がどちらにあるかを探すことに費やされます。

実際の活用例

あるプラットフォームのリポジトリでは、Go、TypeScript、SQL、Python、シェルが同居し、複数言語のモノレポを構成しています。言語ごとにビルド、テスト、リントの方法が異なります。その対策は、入口を一つにすることでした。go-taskの層は、名前付きターゲット578件に拡大し、ルートファイルに50件、その下の名前空間別ファイルに528件を置きました。ネイティブ、Docker、HTTPSの開発モード、コード品質、データベース管理、デプロイを扱います。誰でも `task --list` を実行すれば、ツール一式を見渡せます。ただし、この数は注意すべき点も示します。578件は誰か一人が覚えられる範囲を超えており、使いやすさを保っているのは命名と名前空間による整理です。

別のプロジェクトは2025年11月25日に空のリポジトリから始まり、最初の数週間に決めた三つの方針は、九か月と数千回のコミットを経ても維持されていました。層分けした技術構成、ストアド関数を介したデータアクセス、そして守らせるコードがまだ少ないうちに設けた警告ゼロの基準です。最後の判断の理由こそ、この記事の核心です。コミット5,000件目で導入する基準は整理・改善のプロジェクトになりますが、同じ基準を50件目で導入すれば、それがリポジトリの通常の運用になります。

コードベース内の境界自体が、設計になることもあります。当社の創業者がNextMarinerでアーキテクチャーの一部に貢献した海事プラットフォームは、三層で構成されています。Next.jsのフロントエンドは表示だけを扱います。Fiber上のHumaを使った、意図的に薄いGoのAPIが、ルーティング、検証、処理の取りまとめを行います。業務ロジックはPostgreSQLの関数に置かれます。規則を変えるときは、一つの層、つまりSQLだけを変えます。境界を文書化し、レビューで守らせていました。誰も確認しない慣習は、やがて慣習として機能しなくなるためです。

このサイトのコードベースには、別の種類の境界があります。プロジェクトのページは別のリポジトリで執筆し、スクリプトでこのリポジトリへ書き出してコミットするため、サイトは単独でビルドできます。このコードベース内では、それらのページは生成ファイルです。ここで作業するための指示は、手で編集してはならないと定めています。

問題が起きるところ

コピーしたコードが別々に変化します

二つのアプリケーションで同じ補助処理が必要になり、最も手早い方法としてコピーします。一年後には、異なる修正を加えた二つの版が存在し、一方で直した不具合が、もう一方では残っています。Twelve-Factorは、複数のアプリケーションで同じコードを共有することを規則違反とし、対処方法も示しています。共通部分をライブラリーに移し、依存関係マネージャーを介して取り込み、各コードベースで使用するバージョンを明示します。

秘密情報が履歴に入ります

接続文字列やAPIキーを設定ファイルに入れたままコミットすると、その後はすべてのクローンが持つ履歴に残ります。後のコミットでその行を削除しても変わりません。Twelve-Factorの判定を、コミット前に毎回行う価値があります。このコードベースを今オープンソースとして公開しても、認証情報を一つも漏らさずに済むでしょうか。この判定を通らないものは、それぞれのデプロイ先の環境に置きます。

生成ファイルを手で編集します

スクリプトが書いたファイルも普通のソースファイルに見えるため、誰かが誤字を直接直します。次にスクリプトを実行すると古い文章が書き戻され、修正は何の知らせもなく消えます。生成先のパスを明示し、書き込むのは生成プログラムだけに限定します。修正はすべて、その入力に加えます。

内部の境界が曖昧になります

締め切りに追われ、最短で済むという理由からリクエストハンドラーにクエリを直接書くと、当初の層分けは実際のコードに合わなくなります。一つひとつの近道は小さくても、積み重なると二行の変更がすべての層に及ぶようになります。規則の文書化は役立ち、レビューはさらに役立ちます。ある層から別の層へ踏み込むとビルドが失敗する検査は、最も役立ちます。

基準がコードの半分にしか適用されません

フロントエンド用のリンターだけを用意すると、シェルスクリプトとSQLは検査されず、リンターで防ぐはずだった問題が、そのまま蓄積します。コードベース全体の品質基準には、そこで使うすべての言語を明示する必要があります。これが複数言語を使うリポジトリの、見えにくい負担です。

フォークから別の製品が始まります

チームが関連製品を始めるためにリポジトリをコピーします。両方に同じルートコミットが残るため、Twelve-Factorの規則では、依然として二つのアプリケーションに使われる一つのコードベースです。そして、先に説明した分岐が、リポジトリ全体の規模で始まります。ルートコミットを比較すれば、どちらの状態かが分かります。

```
git -C app-a rev-list --max-parents=0 HEAD
git -C app-b rev-list --max-parents=0 HEAD
```

出力が同じなら、ルートとコードベースを共有しています。異なるなら、別々の二つの履歴です。

参考資料

- I. コードベース — 一つのコードベースから複数のデプロイを行う規則と、その根拠となるルートコミットの共有による判定を説明します。
- II. 依存関係 — すべての依存関係をマニフェストで宣言して分離し、何を取り込むのかをコードベースで厳密に示す方法を説明します。
- III. 設定 — コードベースの外に置くべきものを判断する、オープンソースとして公開できるかという判定を説明します。
- gitsubmodules — それぞれの履歴を保ちながら、あるリポジトリを別のリポジトリに埋め込むGitの仕組みを説明します。
- git-rev-list — 履歴のルートコミットを含め、コミットを列挙するためのリファレンスです。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「コードベースとは？」がお客様の組織にどう当てはまるかをご相談ください。
現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/codebase/