Engineer

# リファクタリング

🛠 外部から見た動作を変えず、コードの構造を変えること

## どのようなものか

リファクタリングは、使う人にとっての動作を変えずに、コードの内部構造を変えることです。名前、関数の境界、モジュールの配置は変えますが、入力、出力、副作用は、すべて以前と同じに保ちます。動詞ではリファクタリングするといい、一回の変更自体も、リファクタリングと呼びます。

Martin Fowlerは、著書Refactoringで用語を定義し、2004年に自身のWebサイトでも、改めて説明しました。名詞としては、外から見た動作を保ちながら、理解しやすく、変更のコストが低いコードにする、ソフトウェアの内部構造への一回の変更です。動詞としては、外から見た動作をずっと保ち、そのような変更を重ねることです。両方が重要です。目的は、次の変更にかかるコストを下げることで、制約は、コードの外側の誰にも、何かが起きたと分からないことです。

Fowlerは、境界を明確にします。refactoring.comで、「リファクタリングは、コードをきれいにすることの別名ではない」と書いています。指すのは、特定の技法です。手順ごとに動作を保たないものも含め、どんな方法であれ、コードを組み替える広い意味の言葉には、再構成を使います。リファクタリングを技法として成り立たせるのは、一回の小ささです。一つ一つは、単独ではほとんど何も変えず、失敗しにくい小さな変換で、そのたびに、システム全体が動きます。長く重ねれば、構造を大きく変えることにもなります。

各手順には、名前があります。著書の第二版に付属するrefactoring.comのカタログは、新しい名前と、それが置き換える以前の名前を並べています。関数の抽出（Extract Function、別名Extract Method）と、その逆の関数のインライン化（Inline Function）、変数のインライン化（Inline Variable）、変数名の変更（Rename Variable）、関数の移動（Move Function）、段階の分離（Split Phase）など、何十種類もあります。共通の名前があれば、開発者は英語なら二語で変更を伝えられ、レビュアーは、既知の手順に照らして確認できます。

概念を判断に生かす

「リファクタリングとは？」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。 3社の間で受け渡す代わりに、1つのチームがアイデアのスケッチから、お客様がログインして使える製品まで担当します。

**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。 GIS地図アプリの開発を主導し売上を10倍に — 10x 地図製品の開発を主導して実現した売上成長

**GIS地図アプリの開発を主導し売上を10倍に** お問い合わせ →

# なぜ重要なのか

成功したソフトウェアは、変わり続けます。新機能や修正が重なり、その一つ一つが、次の変更を少しずつ難しくしがちです。Fowlerの説明では、やがて何かを加えるだけで、苦痛を感じるほど時間がかかります。リファクタリングは、その逆に働きます。追加のたびに、不要な複雑さが増えないよう、コードの形を変えます。見返りを測るのは、コードの見栄えではなく、次の変更のコストです。

そのためFowlerは、プロジェクト計画に現れる独立した仕事ではなく、日常のプログラミングの一部とします。三つのタイミングを挙げています。機能を加える前に、加えやすい形か確認し、違えば先にリファクタリングします。無理に機能を押し込むより、通常は速いとしています。機能が動いた後には、数週間後に戻る人が迷わず追える形へ直します。また、知らないコードを読んで理解する必要があるときは、次の人が同じ場所で苦しまないよう、リファクタリングします。最初のものには、準備のためのリファクタリングという名前があります。Fowlerは、背後の規則をKent Beckのものとして紹介しています。どの変更も、まず、簡単に変更できる形までコードを直し、それから変更します。その最初の構造変更こそが、難しい部分になり得ます。

リファクタリングは、機能の追加にはない方法で、リスクも抑えます。Fowlerは、Beckの、二つの帽子という別のたとえも伝えています。リファクタリングの帽子をかぶっている間は、どの変更も動作を保ち、テストは、ずっと合格します。機能を加えるときは、テストを追加し、既存のテストも失敗するため、緊張もリスクも高くなります。どちらの帽子をかぶっているか分かっている開発者は、テストの失敗の意味も分かります。リファクタリング中なら、直前の手順が間違っていたということです。

技術的負債の大半も、この考え方で返済します。機械学習システムの隠れた負債についてのGoogleの論文は、返済する方法の最初に、コードのリファクタリングを挙げています。ツールも役立ちます。現在は、多くの言語のエディターが、

よく使うリファクタリングを自動化します。Fowlerは、便利でも不可欠ではないとし、使えない場合には、小さい手順と頻繁なテストを頼りにします。

## 実際の活用例

この記事のために書いた、注文の合計を計算し、同じ場所でレシートを出力する関数を例にします。

```
function printReceipt(order) {
  let total = 0;
  for (const line of order.lines) {
    total += line.price * line.quantity;
  }

  // レシートの各行を出力する
  console.log(`Order ${order.id}`);
  console.log(`Total: ${total}`);
}
```

同じ合計を請求書にも表示する、という新しい要件が来ました。ループを複製する代わりに、まずリファクタリングします。最初は、関数の抽出です。ループを独立した関数へ移し、計算するものを名前にして、テストを実行します。次に、出力も同じように抽出し、コメントを関数名で置き換え、再びテストを実行します。三つ目は、変数のインライン化です。`total`変数を使うのは一回だけになったため、その場所を関数の呼び出しにします。各手順の後も、プログラムは以前とまったく同じことをし、一つずつ独立してコミットできます。

```
function printReceipt(order) {
  printLines(order.id, orderTotal(order));
}

function orderTotal(order) {
  let total = 0;
  for (const line of order.lines) {
    total += line.price * line.quantity;
  }
  return total;
}

function printLines(id, total) {
  console.log(`Order ${id}`);
  console.log(`Total: ${total}`);
}
```

ここで初めて機能を加えます。請求書のコードに、`orderTotal`を呼ぶ一行を加えるだけです。三つのリファクタリングの手順には、新しいテストは不要でした。外から観測できることは何も変わらず、既存のレシート用のテストが、それを証明するためです。

インフラのコードもリファクタリングしますが、名前が稼働中のリソースを指すため、失敗の影響は、より大きくなります。Terraformのドキュメントは、設定内でリソースへ新しいアドレスを与えると、既定では、既存のものを破棄し、新しく構築する指示になると警告します。そのため、読みやすくするつもりで`aws_instance.a`を`aws_instance.web`へ変えると、サーバーを置き換えます。Terraform v1.1以降は、`moved`ブロックで以前のアドレスを記録でき、次の計画は、何も破棄せず、アドレスを更新します。

```
resource "aws_instance" "web" {
  # （設定は変更なし）
}

moved {
  from = aws_instance.a
  to   = aws_instance.web
}
```

それより前のバージョンでは、ドキュメントは、同じ移動に`terraform state mv`コマンドを案内しています。どちらでも、計画を読み、破棄と作成ではなく、移動を示していることを確かめて初めて、安全なリファクタリングになります。

履歴に変更の種類が残れば、こうした変更を機能追加から分けやすくなります。このサイトの四つのリポジトリーに共通の履歴規約を適用した事例で説明する規約では、すべてのコミットメッセージを、変更の種類を示す所定の接頭辞で始めます。コミットメッセージ用のフックが、この規約を強制します。2026年7月の作業を扱うページでは、最大のリポジトリーの履歴に、26件のリファクタリング、156件の機能追加、133件の修正を数えています。同じページは、フックが検査するのは書式で、内容ではないとも率直に述べています。リファクタリングと名付けたコミットでも、動作を保ったことは、テストとレビューで示す必要があります。

# 問題が起きるところ

**リファクタリングに、動作の変更を混ぜます。** 関数を整えると同時に、そのバグも同じコミットで直すと、テストが失敗しても、どちらが原因か分からず、レビュアーは、どの行も、動作を変える可能性があるものとして読む必要があります。一度にかぶる帽子は一つにします。一つのコミットでリファクタリングし、次で動作を変えます。途中でテストが失敗したら、修正を重ねて先へ進むのではなく、直前の手順を取り消します。

**大規模な再構成を、リファクタリングと呼びます。** コードを「リファクタリング中」として、何週間もビルドが壊れたままのブランチがあります。Fowlerの言葉では再構成であり、リファクタリングが避けるためにあるリスクを抱えます。システムが動いたままで、個別にマージできる手順へ分けます。それができなければ、書き直しと呼び、その前提で計画します。

**変わらなかったことを示すテストがありません。** 動作を保ったというのは主張であり、テストがなければ推測です。テストのないコードへ触る前に、おかしく見える点も含め、現在の動作を固定するテストを書き、奇妙な点の修正は、後で別の変更にします。準備のためのリファクタリングの例にある、Fowlerの目安も当てはまります。コードを実行し、目で出力を確かめたくなったら、代わりにテストを書きます。

**インフラの名前変更で、リソースが置き換わります。** Terraformの分かりやすい新しい名前は、レビューではリファクタリングに見えても、計画は、データベースの破棄と、空のものの作成を静かに提案します。名前の変更と、モジュールへの出入りを伴う移動は、すべて `moved` ブロックへ記録します。リファクタリングのつもりの計画に、破棄があれば、失敗した手順として扱います。

**古い `moved` ブロックを削除します。** ブロックが増えると、削除が整理に見えます。Terraformのドキュメントは、破壊的変更としています。以前のアドレスをまだ使う設定では、移動ではなく、オブジェクトを破棄する計画ができるためです。履歴のブロックを保ち、二度改名したら、つなげます。削除できるのは、非公開のモジュールで、すべての利用者が、ブロックを含むバージョンを適用済みになった後だけです。

**専用の時期を設けるまで、リファクタリングを待たせます。** 後で「リファクタリングのスプリント」を予定するチームは、変更しにくいコードへ機能を追加し続けます。後のスプリントは、締め切りが動けば、簡単に削られます。Fowlerの助言は逆です。各変更の中で、まず変更を簡単にするために、その後は結果を明快に保つためにリファクタリングし、計画に専用の枠が必要な仕事にしません。

# 参考資料

- リファクタリングの定義 — 著書で示す、Martin Fowlerの名詞と動詞の定義です。
- Refactoring — 日常のプログラミングに含める理由と、完全なカタログを載せる、Fowlerの著書のサイトです。
- 関数の抽出 — よく使うリファクタリングの一つで、逆の操作と、初版での名前も載せています。
- 準備のためのリファクタリングの例 — Fowlerが、機能追加の前に小さい手順で自分のコードを整え、二つの帽子を説明します。
- モジュールのリファクタリング — `moved` ブロック、名前変更、モジュールの分割、古いブロックを残す理由についてのTerraformのドキュメントです。

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

「リファクタリングとは？」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/refactoring/