

リファクタリング  
既存のコードを安全に改善する

# Refactoring

## Improving the Design of Existing Code

[著] Martin Fowler

[共訳] 児玉公信・友野晶夫・平澤章・梅澤真史

# 目次

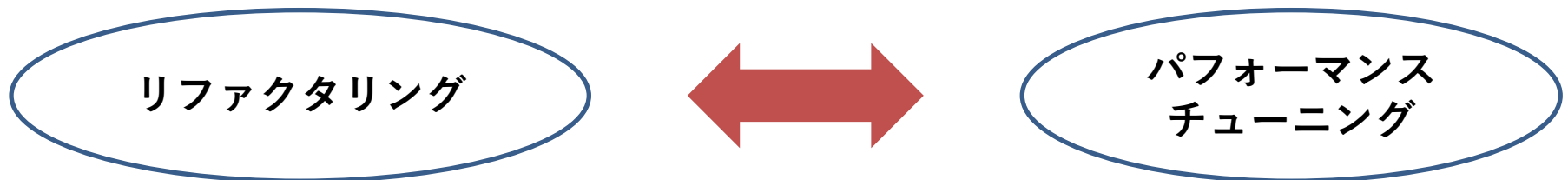
1. リファクタリングー最初の例
2. リファクタリングの原則
3. コードの不吉な臭い
4. テストの構築
5. リファクタリング・カタログに向けて
6. メソッド抽出
7. オブジェクト間での特性の移動
8. データの再編成
9. 条件記述の単純化
10. メソッド呼び出しの単純化
11. 継承の取り扱い
12. 大きなリファクタリング
13. リファクタリング，再利用，現実
14. リファクタリングツール
15. 部品から全体へ

# リファクタリングの原則

- リファクタリングとは何であるのか
- なぜリファクタリングすべきか
- リファクタリングの重要な原則
- リファクタリングする上で考慮する点

# リファクタリングの定義

1. ソフトウェアを理解しやすく，変更を容易にするために内部構造を変更する.



2. ソフトウェアの外的振る舞いを保つ.  
開発者やユーザには気付かれない.

# リファクタリングを行う理由

- ソフトウェアの設計の改善
  - 適切でない部分を取り除きコードを整理する。  
重複コードを排除することで設計が良くなる。
- ソフトウェアを理解しやすくする
  - 将来の開発者に向けて、うまく構造化したコードによって目的を伝える。
- より速くプログラミングを行う
  - 設計の劣化を防ぎ、理解しやすくすることで、ラピッドな開発が可能となる

# いつリファクタリングすべきか (1)

- 3ヶ月に1回, 2週間~~に~~リファクタリングする
- 全体の作業の中で自然に発生するもの
  - リファクタリングしようと構えるものでない
- 3度目の法則 (Don Robert)
  - 同じ作業を3度していることに気付いた時

# いつリファクタリングすべきか (2)

- 機能追加の時
  - 機能追加の際, コードを理解するために
  - 容易に機能追加できない設計を直す
- バグフィックスの時
  - コードを洗練させることで, バグを見つける
- コードレビューの時
  - レビュー時に浮んだアイデアを適用できるか
  - レビューによる意見が実装しやすい

# リファクタリングの問題点

- データベース
  - データベーススキーマとオブジェクトモデルとの依存性が高い
  - コードの変更によりデータベーススキーマを変更する必要がある
- リファクタリングしにくい設計
  - 間違った設計の場合、リファクタリングが困難
  - 時間をかけて設計から見直す

# リファクタリングと設計

- 設計を補完する役割がある
- 事前の設計が完璧であると思う必要は無い
  - コードを作り込む内に事前の設計が考えていたことと違うと気付く
  - 事前の設計を維持したまま変更しようとする  
と複雑になり，保守がしにくい
- リファクタリングにより設計を簡素化することで，比較的簡単に解決できる

# リファクタリングとパフォーマンス

- 理解しやすくする変更は実行速度を落してしまいがち
- 反面、パフォーマンスチューニングしやすい形にすることもできる
  - 該当プログラムを見つけやすくなる
  - チューニングに十分時間をかけれる
- リファクタリングが速いソフトウェアを作るのに重要な技術である