

Why We Refactor ?

Danilo Silva, Nikolaos Tsantalis , Marco Tulio Valente

発表者 森岡 勇介

はじめに

- リファクタリングとは、ソフトウェア開発において、プログラムのふるまいや動作を変えずに、内部の設計や構造を見直し書き換えること。
- コードの保守性、読みやすさを改善するのに役立つ。
- 特定のリファクタリングの実際の動機を経験的に調査した研究は限られている。

目的

- リファクタリングの適用を促進する動機のリストを作成する.
- 作成したリストによって, 研究者や実務家がプロジェクトのコミット履歴を分析するときに同期の自動検出をするためのルールを推測するのに役立つようにする.

目的

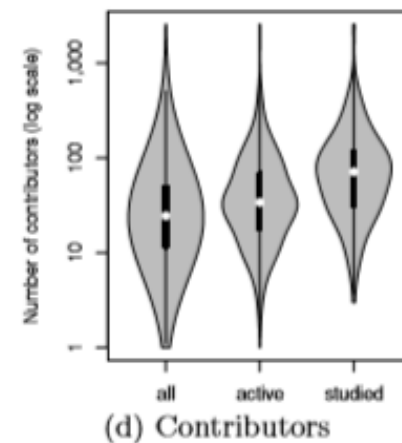
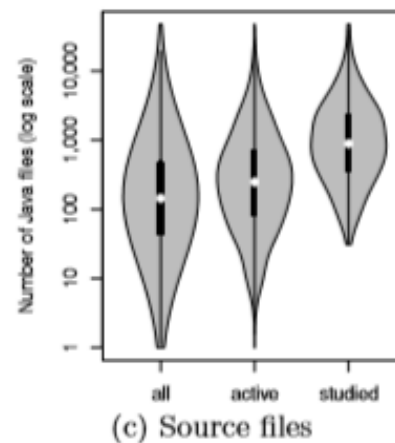
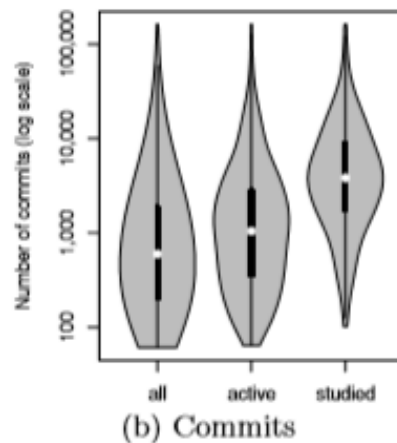
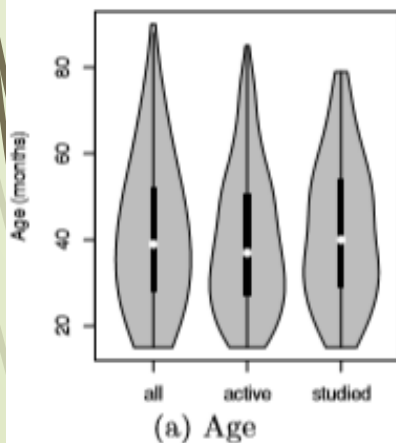
リファクタリングには設計の改善，コードの理解，バグの発見，生産性の向上などの目的がある．

しかし，リファクタリング推奨システムの研究ではcode smellの解決に焦点が合わせられたものが多い．

開発者の需要とプラクティスに沿ったリファクタリング推奨システムの開発の手助けになる．

分析するリポジトリ

- GitHubの中から人気順に並べたときの上位1000個のJavaリポジトリを選択し、それらをコミットの多い順位並び替え、下位4分の1を破棄したもの。



RefactoringMiner Tool

- 今回の研究に使用したリファクタリングを検出する道具.
- 12個の有名なリファクタリングタイプを検出する.
- クラス/メソッド/フィールドの名前変更リファクタリングの検出はできない.
- RefactoringMinerにリファクタリングの検出を依存しているので, 精度が大事.

EXTRACT METHOD
MOVE CLASS
MOVE ATTRIBUTE
RENAME PACKAGE
MOVE METHOD
INLINE METHOD
PULL UP METHOD
PULL UP ATTRIBUTE
EXTRACT SUPERCLASS
PUSH DOWN METHOD
PUSH DOWN ATTRIBUTE
EXTRACT INTERFACE

RefactoringMiner Tool の 精度

Table 1: RefactoringMiner Recall and Precision

Refactoring	TP	FP	FN	Recall	Prec.
EXTRACT METHOD	15	2	0	1.00	0.88
INLINE METHOD	13	0	2	0.87	1.00
PULL UP ATTRIBUTE	15	0	0	1.00	1.00
PULL UP METHOD	12	0	3	0.80	1.00
PUSH DOWN ATTRIBUTE	15	0	0	1.00	1.00
PUSH DOWN METHOD	13	0	2	0.87	1.00
MOVE ATTRIBUTE	15	0	0	1.00	1.00
MOVE METHOD	13	0	2	0.87	1.00
Total	111	2	9	0.93	0.98

分析方法

- 61日間リポジトリを監視し、RefactoringMinerでリファクタリングが検出されると質問を投げかけ、協力してもらう。
- スпамメールだと思わせないために、同じ人に再度質問することはしていない。
- コミットの説明にリファクタリングの説明がなされている場合、質問はしていない。
- 得られた回答をもとにリファクタリングを行う動機を分析する。

分析方法

- 195件の回答を得て、27件質問を行わず情報を取得している。
- 投げかけた質問は以下の二つです。
 1. リファクタリングを実行した理由を説明してください。
 2. IDEの自動リファクタリングサポートを使用してリファクタリングを事項しましたか。

なぜ開発者はリファクタリングを行ったのか

➡ 抽出メソッド

Table 3: EXTRACT METHOD motivations

Theme	Description	Occurrences
Extract reusable method	Extract a piece of reusable code from a single place and call the extracted method in multiple places.	43 ■■■■
Introduce alternative method signature	Introduce an alternative signature for an existing method (e.g., with additional or different parameters) and make the original method delegate to the extracted one.	25 ■■
Decompose method to improve readability	Extract a piece of code having a distinct functionality into a separate method to make the original method easier to understand.	21 ■■
Facilitate extension	Extract a piece of code in a new method to facilitate the implementation of a feature or bug fix, by adding extra code either in the extracted method, or in the original method.	15 ■■
Remove duplication	Extract a piece of duplicated code from multiple places, and replace the duplicated code instances with calls to the extracted method.	14 ■■
Replace Method preserving backward compatibility	Introduce a new method that replaces an existing one to improve its name or remove unused parameters. The original method is preserved for backward compatibility, it is marked as deprecated, and delegates to the extracted one.	6 ■
Improve testability	Extract a piece of code in a separate method to enable its unit testing in isolation from the rest of the original method.	6 ■
Enable overriding	Extract a piece of code in a separate method to enable subclasses override the extracted behavior with more specialized behavior.	4 ■
Enable recursion	Extract a piece of code to make it a recursive method.	2
Introduce factory method	Extract a constructor call (class instance creation) into a separate method.	1
Introduce async operation	Extract a piece of code in a separate method to make it execute in a thread.	1

なぜ開発者はリファクタリングを行ったのか

Move Class
Move Attribute
Rename package

Type	Theme	Description	Occurrences
MC	Move class to appropriate container	Move a class to a package that is more functionally or conceptually relevant.	13 
MC	Introduce sub-package	Move a group of related classes to a new subpackage.	7 
MA	Move attribute to appropriate class	Move an attribute to a class that is more functionally or conceptually relevant.	15 
MA	Remove duplication	Move similar attributes to another class where a single copy of them can be shared, eliminating the duplication.	4 
RP	Improve package name	Rename a package to better represent its purpose.	8 
RP	Enforce naming consistency	Rename a package to conform to project's naming conventions.	3 

自動リファクタリングサポートを使用したか

自動より手動でリファクタリングを行っている数のほうが多い。

Table 5: Manual vs. automated refactoring

Modification	Occurrences	
Manual	95	
Automated	66	
Not answered	48	
Partially automated	13	

なぜ手動で行うのか

- 29人の開発者になぜ手動で行うのか説明してもらった。

Table 7: Reasons for not using refactoring tools

Description	Occurrences
The developer does not trust automated support for complex refactorings.	10
Automated refactoring is unnecessary, because the refactoring is trivial and can be manually applied.	8
The required modification is not supported by the IDE.	6
The developer is not familiar with the refactoring capabilities of his/her IDE.	3
The developer did not realize at the moment of the refactoring that he/she could have used refactoring tools.	2

考察

- 抽出メソッドはcode smell解決のためにリファクタリングを行っていた.
- しかし, 実際の動機はcode smellによるものでないものが多い.
- 研究者等が**他のリファクタリング**をサポートするリファクタリング推奨システムを設計するようになると考えられる.

外部への有効性

- オープンソースのJavaベースのGitHubでホストされるプロジェクトに限定されている.
- したがって、調査結果が**産業用システム、ほかの言語で実装されたシステム**に適用されると主張できない.

まとめ

- Code smellの解決によるリファクタリングは数が少ない。抽出メソッドの11の動機のうち2つのみ。
- 抽出メソッドは複数の目的、特にコードの再利用と機能拡張に関連する目的に役立つ重要な操作。テスト用遺制、パブリックAPIメソッドを廃止するためにも使用される。
- 手動リファクタリングは依然として普及している。
- リファクタリング推奨システムの今後の研究は焦点をcode smell関連のものより**機能拡張のしやすさ**などに合わせるべきである。