

自動生成されたパッチが コーディング規約違反に 与える影響の分析

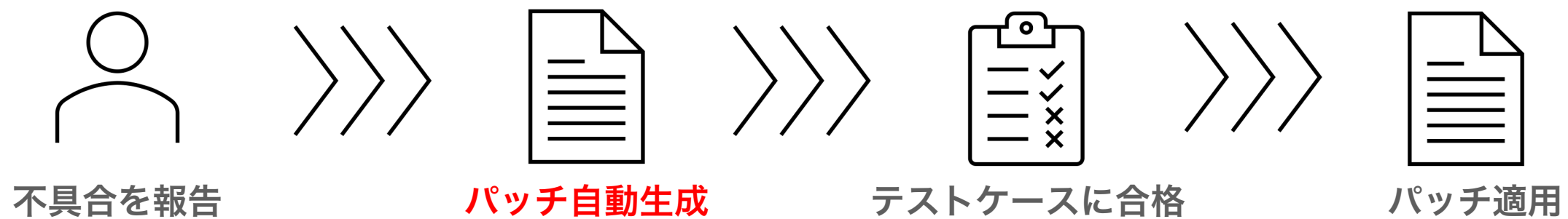
オープンソースソフトウェア工学研究室
野口 優輝

背景

自動プログラム修正 (Automated Program Repair)

- プログラムを修正するためのパッチを自動で生成する技術

自動プログラム修正の流れ



→開発者の不具合修正における労力を減らすことができる

自動生成パッチ (APRパッチ)

- APRツールによって自動生成されたパッチ
- 開発者が意図する機能を崩すことなく不具合を修正することが期待されている^[1]

[1] Durieux, T., Martinez, M., Monperrus, M., Sommerard, R., and Xuan, J.: Automatic repair of real bugs: an experience report on the defects4j dataset, Empirical Software Engineering, 2017, pp1936-1964

APRパッチの品質

パッチの正しさはテストケースに合格するかで保証

- プログラムの振る舞いにおいての正しさは保証される
- コードの書き方が正しいかどうかは保証できない

パッチが生成したコードの品質

- コーディング規約に従ったコードは可読性や保守性を保ち、コードの品質を高めることができる^[2]

→APRパッチもコードの品質を考慮することは重要

コーディング規約に沿ったパッチを生成する必要がある

[2]織田 泰輔, 阿萬 裕久, 佐々木 隆志, 川原 稔, オープンソース開発でのコード修正におけるコーディング規約違反の変化に関する調査, 情報処理学会研究報告会2014, pp1-8

コーディング規約

ソフトウェアにおけるコードの書き方のガイドライン

- 各プロジェクトに独自のルールが設定されている

静的解析ツールで自動検査

- Javaでは`CheckStyle`によりコーディング規約違反を検出^[3]

関連研究

- 静的解析ツール使用による違反の特定は、目視より容易であり、コーディング規約遵守に有効である^[4]

APRパッチがコーディング規約違反に
与える影響を調べた研究は存在しない

[3] Beller, M., Bholanath, R., McIntosh, S., and Zaidman, A.: Analyzing the state of static analysis: a large-scale evaluation in open source software, Proc. SANER, 2016, pp. 470–481.

[4] Panichella, S., Arnaoudova, V., Di Penta, M., and Antoniol, G.: Would static analysis tools help developers with code reviews?, Proc. SANER, 2015, pp. 161–170.

目的とリサーチクエスチョン (RQ)

目的

APRパッチがコーディング規約違反に与える影響を明らかにする

RQ

RQ1：規約違反数を増加させるAPRパッチはどの程度か？

RQ2：APRパッチ適用で発生する規約違反に特徴はあるか？



開発者パッチ（人間が作成したパッチ）と比較する

データセット

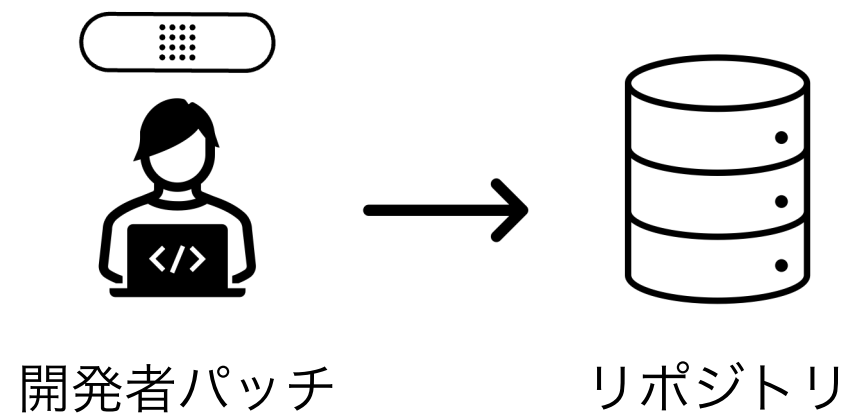
- [5]の内, CheckStyleを利用するプロジェクト
 - Apache Commons Lang
 - Apache Commons Math
 - Joda Time
- 開発者が作成したパッチ
- 32種類のAPRツールが生成したパッチ

パッチ生成元	Lang	Math	Time	パッチ生成元	Lang	Math	Time
開発者 [12]	105	184	53	jGenProg2015 [5]	0	4	0
3sFix [4]	0	2	0	jKali [23]	2	1	0
ACS [36]	3	15	1	jMutRepair [23]	0	1	0
ARJA [39]	6	10	1	KaliA [39]	0	1	0
CapGen [33]	7	14	0	kPAR [18]	1	4	0
Cardumen [24]	0	1	0	LSRepair [20]	4	1	0
CoCoNut [22]	7	17	1	Nopol2015 [5]	3	0	0
ConFix [14]	6	9	1	PraPR [7]	5	10	3
DeepRepair [34]	4	2	0	RSRepairA [39]	0	0	0
DynaMoth [6]	0	1	0	SequenceR [3]	2	8	0
ELIXIR [27]	8	12	2	SimFix [10]	9	16	1
FixMiner [15]	0	8	0	SketchFix [8]	2	6	0
GenPat [9]	4	3	0	SOFix [21]	3	13	1
GenProgA [39]	0	0	0	ssFix [35]	5	7	0
HDRRepair [16]	1	5	1	Tbar [19]	7	8	0
Hercules [28]	13	30	8	Vfix [37]	7	0	0
JAID [2]	13	11	0	APR パッチ合計	122	220	20
				All (開発者+ APR)	231	434	75

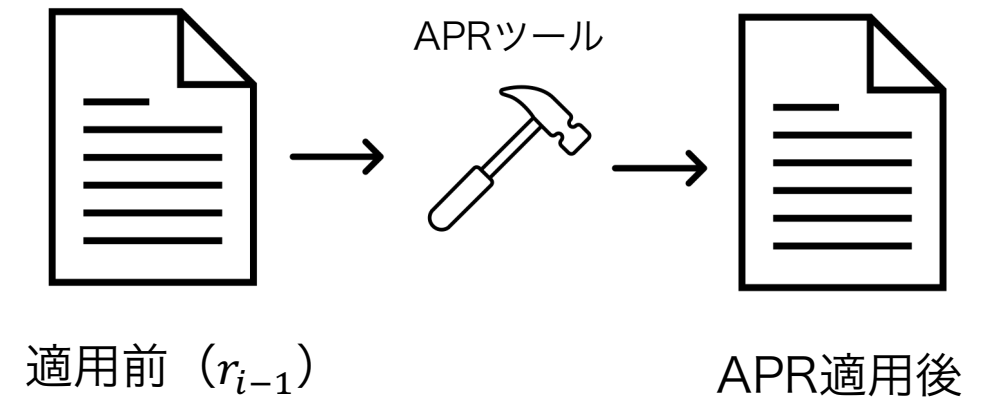
[5] Tian, H., Li, Y., Pian, W., Kaboré, A. K., Liu, K., Habib, A., Klein, J., and Bissyandé, T. F.: Predicting patch correctness based on the similarity of failing test cases, Trans. Softw. Eng. Methodol., Vol. 31, No. 4(2022), pp. 1-30.

分析の流れ

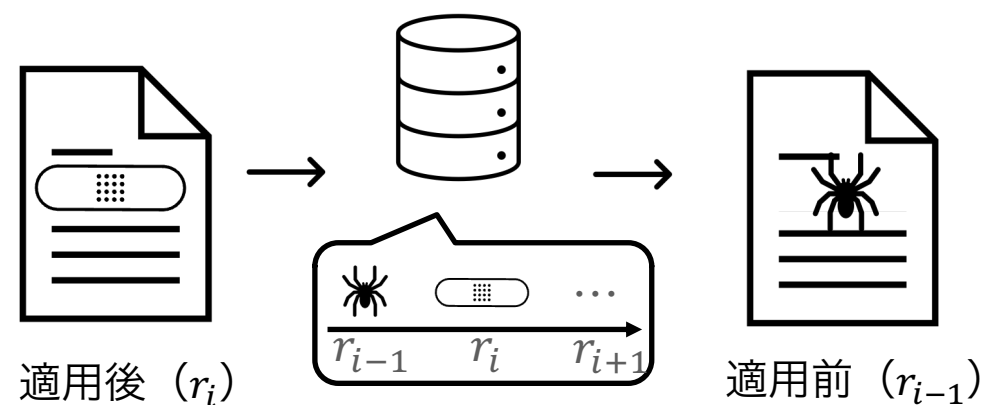
1. 開発者パッチ適用後のファイル検索



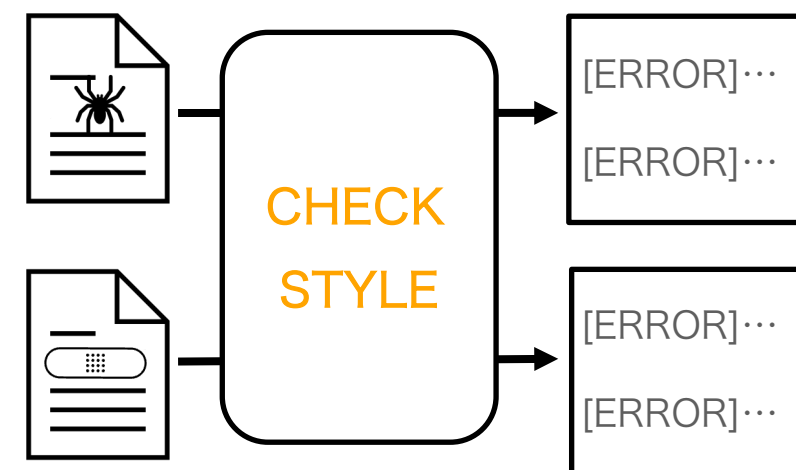
3. APRパッチの適用



2. 開発者パッチ適用前後ファイル特定



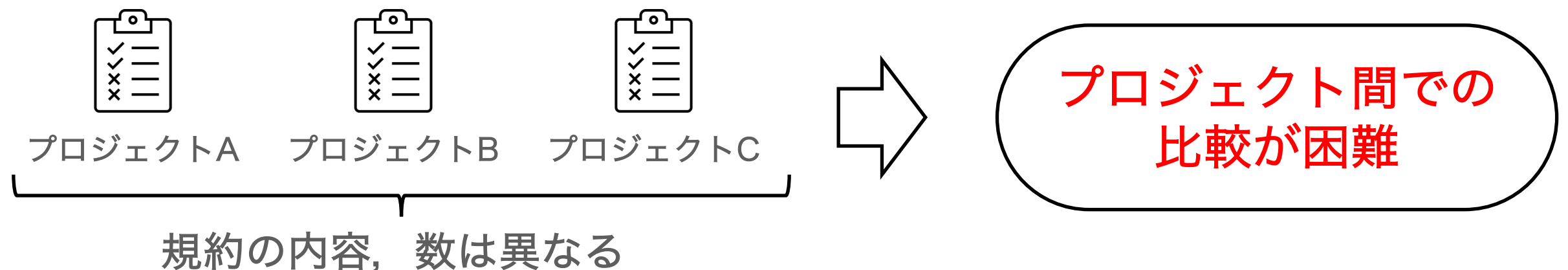
4. CheckStyleの実行



使用するCheckStyleファイル

projectスタイル

- プロジェクト独自で定義されたCheckStyleファイル
- プロジェクトごとに規約の内容や数が異なる

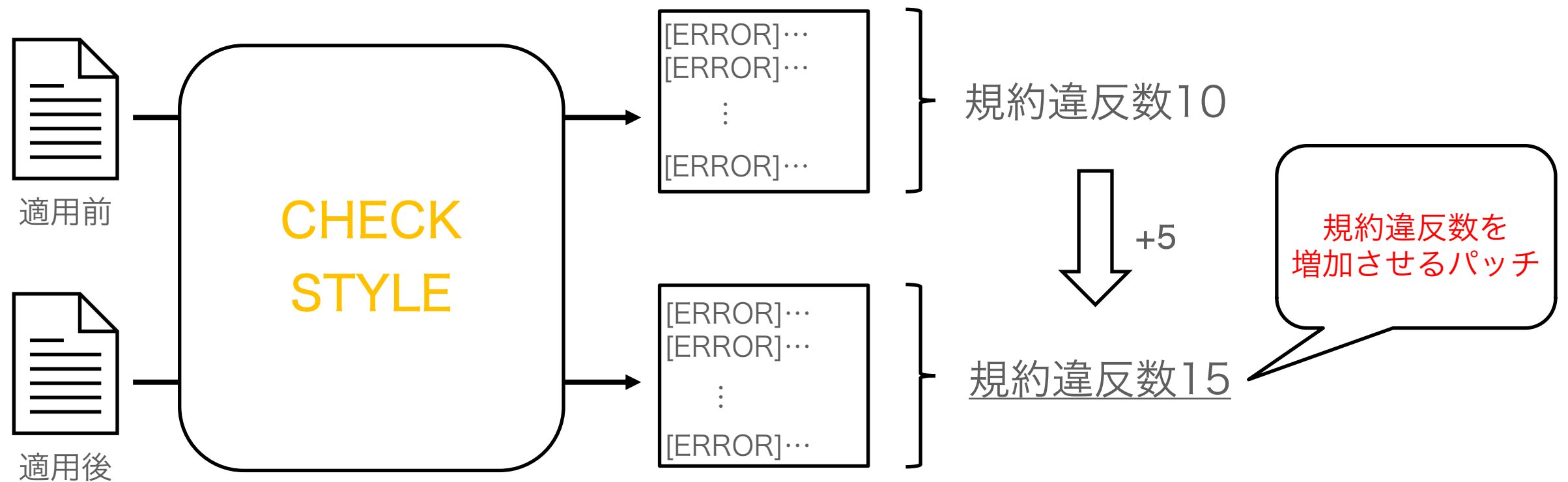


Googleスタイル, Sunスタイル

- 一般的に広く参照されているCheckStyleファイル
- projectスタイルを用いるよりも, 規約違反数の変化をよく捉えることができる

RQ1：分析方法

APR・開発者それぞれのパッチ適用による規約違反数の変化を分析



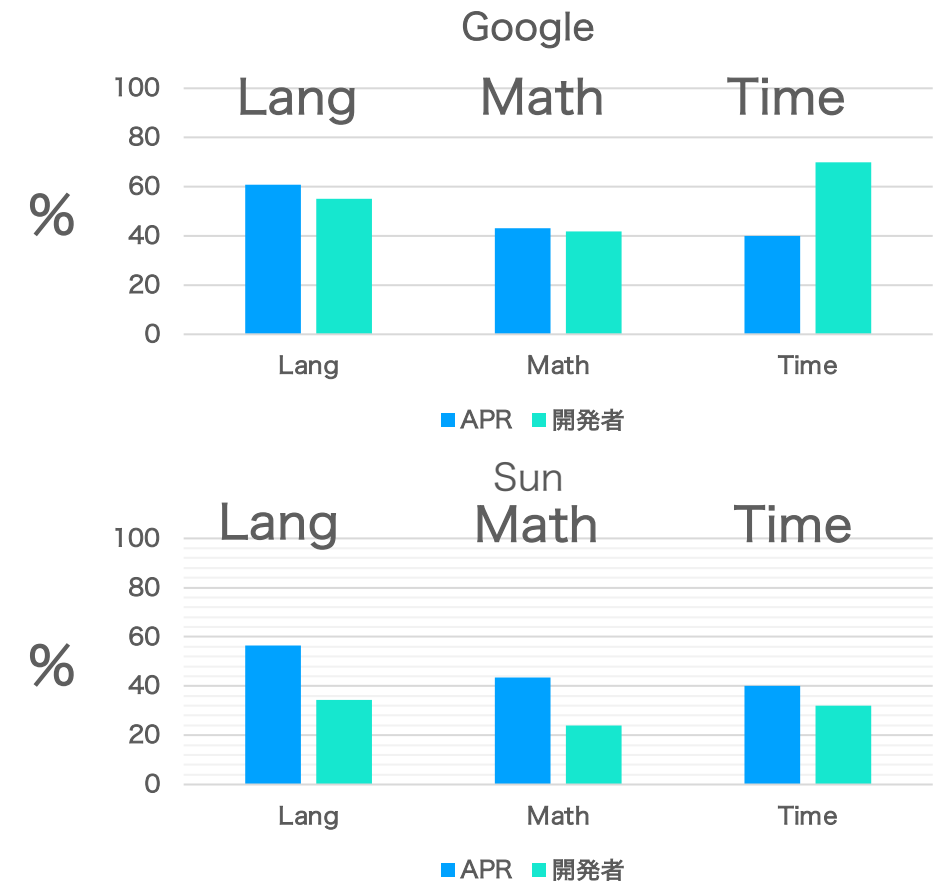
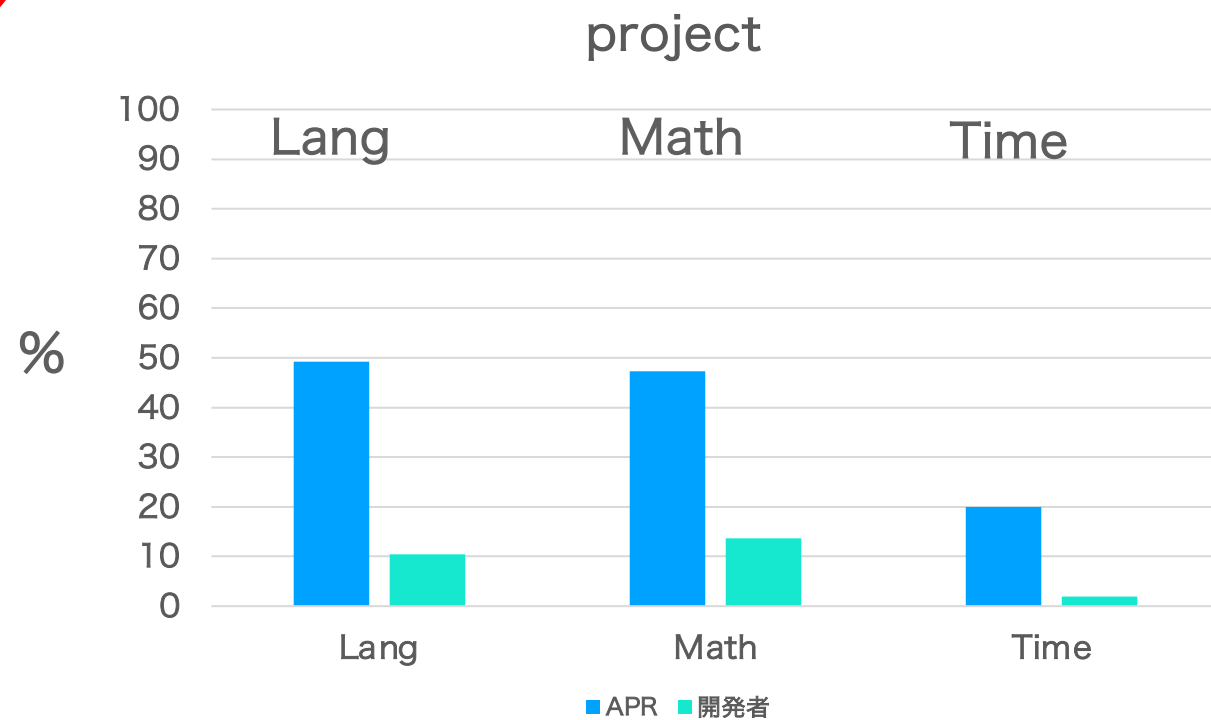
計算方法

$$\text{割合(\%)} = \frac{\text{規約違反数が増加したパッチの個数}}{\text{プロジェクトにおけるパッチの個数}} \times 100$$

(増加)

RQ1：規約違反数を増加させるパッチはどの程度か

RQ1：分析結果



全てのスタイルで，APRは規約違反数を増加させるパッチの割合が高い

- projectスタイルでは，特に開発者パッチと比較しても，割合が高くなっていることがわかる

RQ2：分析方法

Google, Sunスタイルを用いて分析

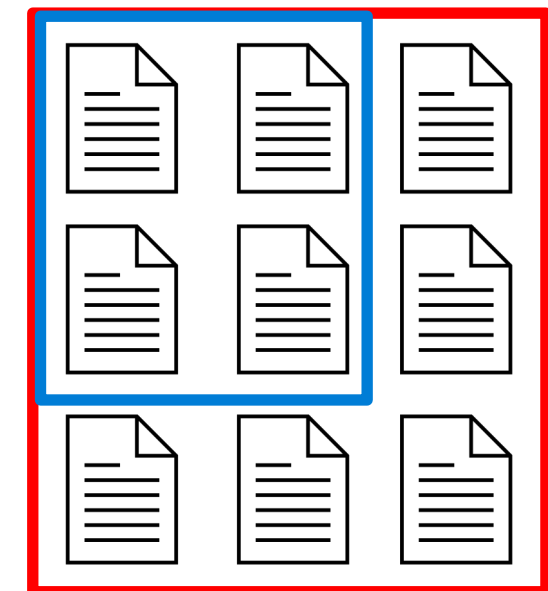
- projectスタイルでは、プロジェクトごとにルールが異なる
→APRが発生させやすい違反の特徴が定まらない

分析手順

- パッチ適用で発生した規約違反を集計
- 規約違反ごとにパッチの割合を計算

計算方法

$$\text{割合} = \frac{\text{ある違反が発生したパッチの総数}}{\text{違反数が増加したパッチの総数}} \times 100$$



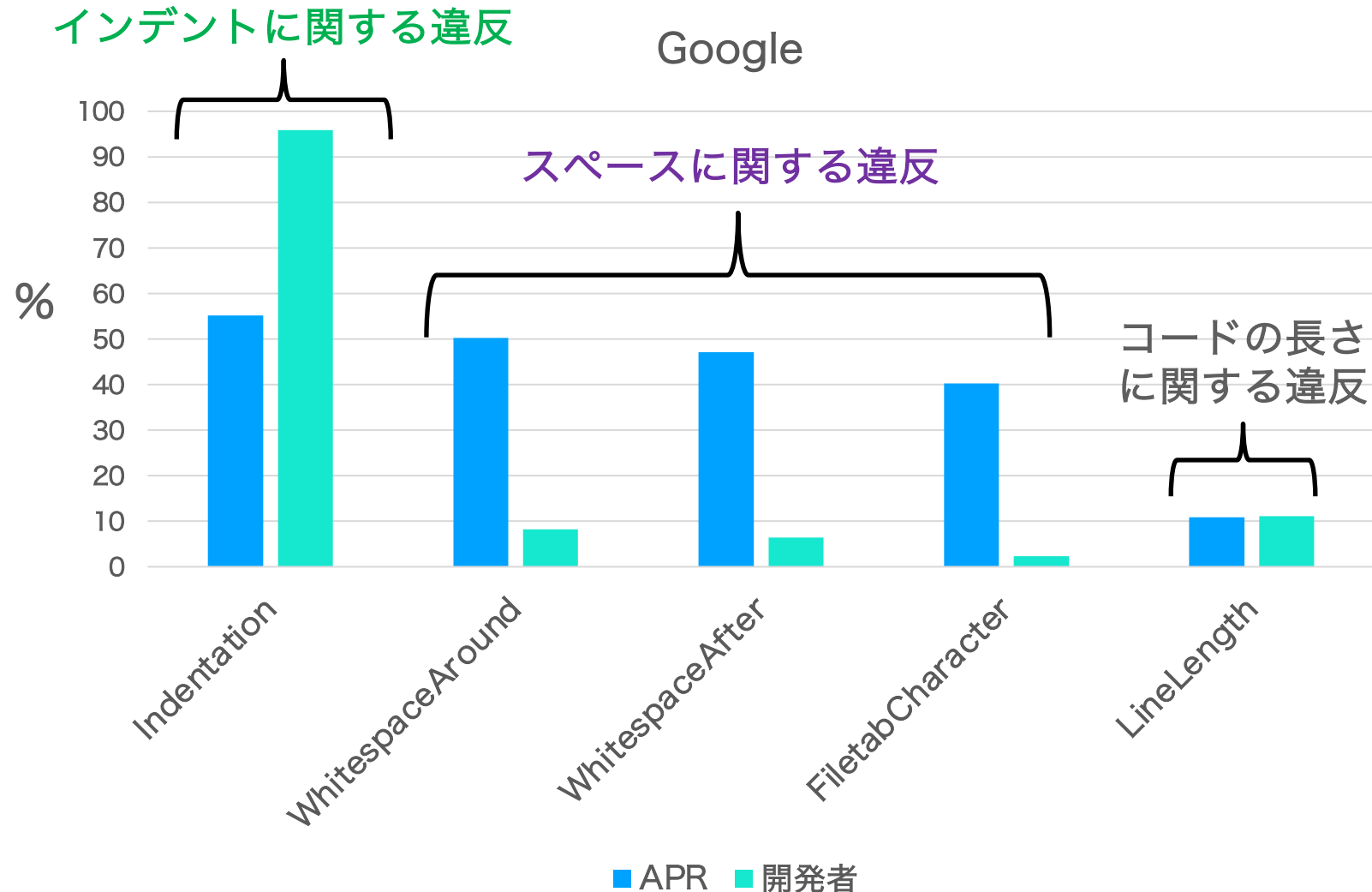
違反Aが発生

$$\frac{4}{9} \times 100 = 44.44...$$

約44%, 違反Aが発生している

RQ2：発生する違反の特徴は？

RQ2：分析結果 (Google)



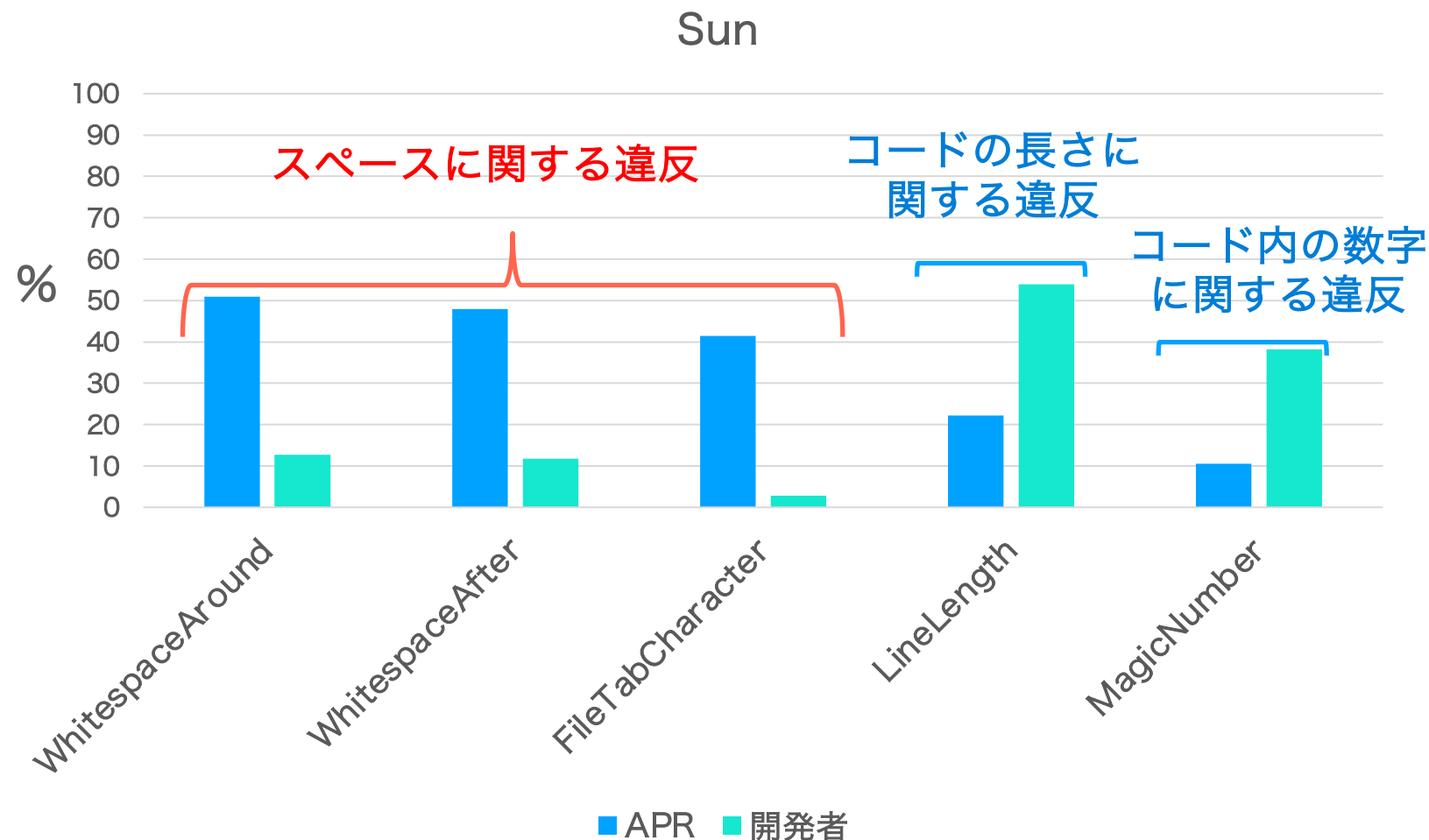
違反名	説明
Indentation	インデントがルールに合致
Whitespace Around	トークンの前後に空白があるか
Whitespace After	特定の文字の後に空白があるか
FileTab Character	タブ文字ではなく空白を使ってるか
LineLength	1行あたりにある文字数

APRは開発者と同様にインデントの違反が多い

APRはスペースに関する違反が多い

RQ2：発生する違反の特徴は？

RQ2：分析結果 (Sun)



違反名	説明
WhitespacesAround	トークンの前後に空白があるか
WhitespaceAfter	特定の文字の後に空白があるか
FileTabCharacter	タブ文字ではなく空白を使ってるか
LineLength	1行あたりにある文字数
MagicNumber	マジックナンバーを使用しているか

Google同様, APRはスペースに関する違反が多い

APRは開発者と比較して, コードの長さ, マジックナンバーに関する違反は少ない

WhitespaceAfterの規約違反の例

APRパッチはスペースに関する違反が多い

開発者パッチ適用後

```
public static float max(final float a, final float b) {  
    return (a <= b) ? b : (Float.isNaN(a + b) ? Float.NaN : a);  
}
```

スペースあり

APRパッチ適用後

```
public static float max(final float a, final float b) {  
    return( a <= b ) ? b :( Float.isNaN( a + b ) ? Float.NaN : a ) ;  
}
```

スペースなし

WhitespaceAfterの違反発生

APRツールにコード規約遵守の観点を取り入れる必要がある

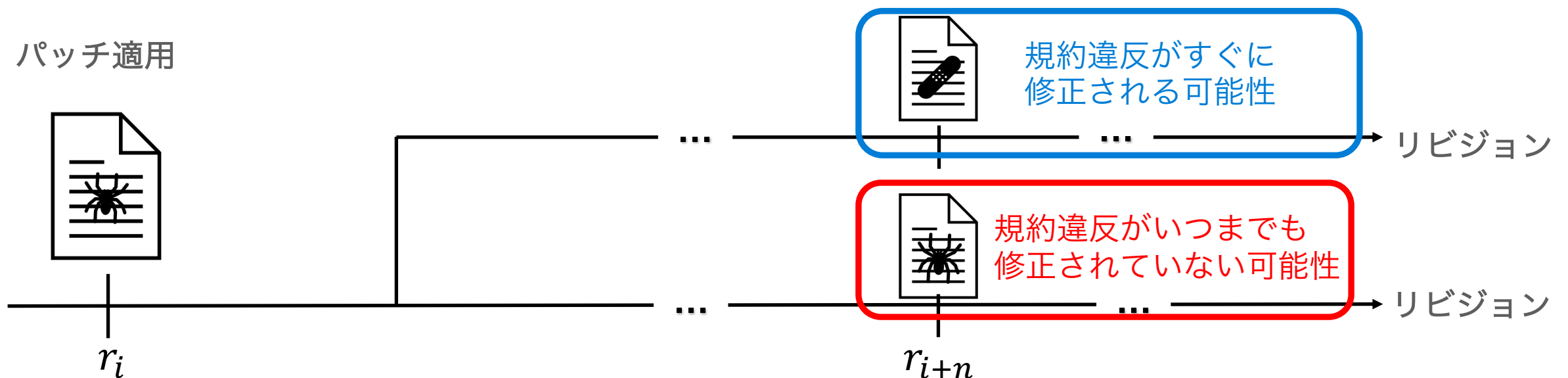
考察

APRは規約違反数を増加させるパッチの割合が高く、スペースに関する違反を多く発生させた

- APR技術はコードのトークン化や抽象構文木化を利用していることで、コードのスタイルが失われやすい

疑問点

- スペースなどの違反はIDE（統合開発環境）で自動修正可能？
→コーディング規約を考慮したパッチ生成は不要？



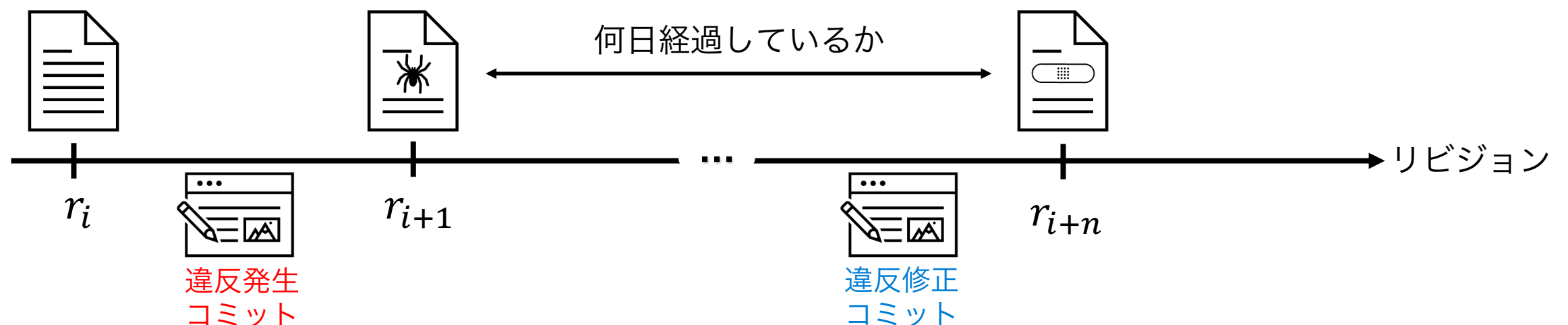
追加分析

目的

- パッチ適用後のリビジョンで、規約違反が修正されているかを明らかにする

アプローチ

- パッチ適用で発生した違反行を含むコミットを探索
- 違反修正コミットがある場合、修正までにかかった日数を計測



分析対象

projectスタイルで違反が発生した開発者パッチ

プロジェクト	パッチ(個)	規約違反(個)
Lang	13	13
Math	41	51

APRパッチは対象にできない

- APRパッチは実際のプロジェクトでは使用されていないため、実際に適用された開発者パッチのみを使用する

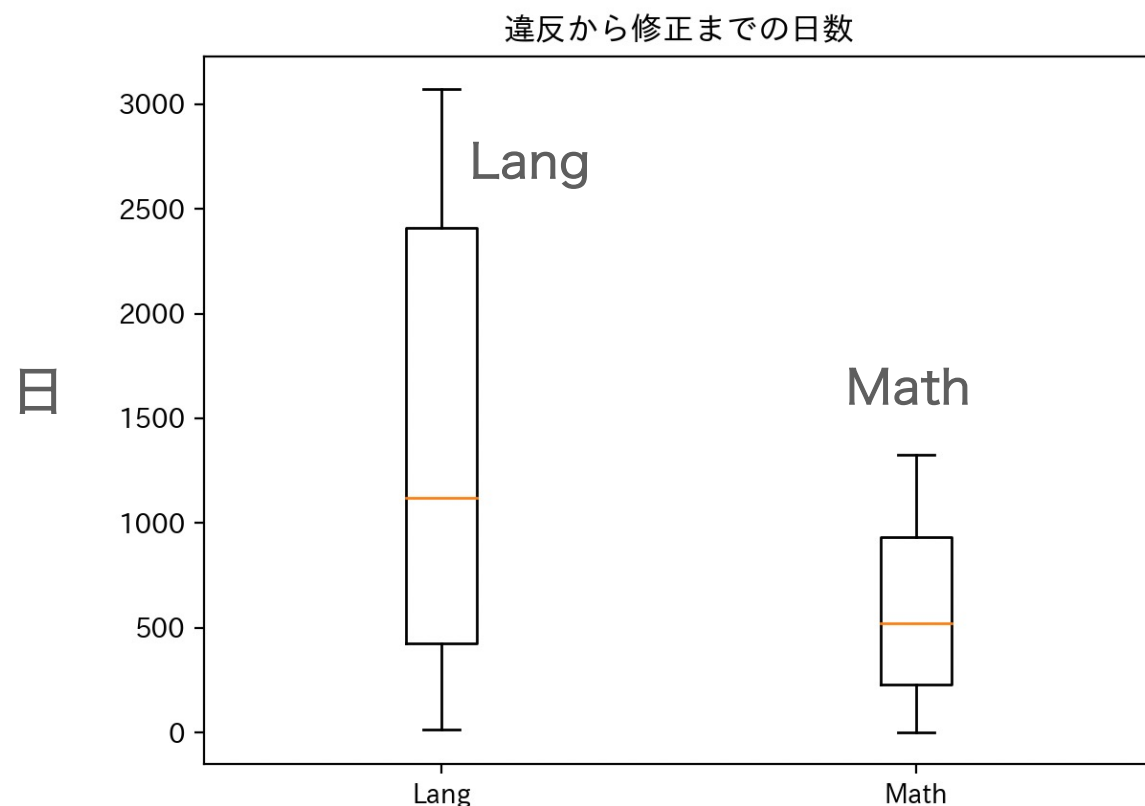
開発者パッチでの違反が修正されない場合、
APRパッチを用いた場合でも修正されない可能性

分析結果

Lang, Mathともに大部分の違反が修正されていた

- Langでは, 11個/13個が修正
- Mathでは, 30個/51個が修正

修正までには相当な日数を要していた



プロジェクト	平均値(日)	中央値(日)
Lang	1430	1120
Math	978	522

違反から修正までの日数

総括

目的

- APRパッチを適用することで、発生するコーディング規約違反について分析

結果

- APRは規約違反数を増加させるパッチの割合が高い
- APRパッチはスペースに関する違反を多く発生
- パッチ適用で発生する規約違反は長い間放置される

投稿論文

野口 優輝, 木村 祐太, 伊原 彰則, 大平 雅雄, "自動生成されたパッチがコーディング規約違反に与える影響の分析", FOSE2023 第30回ソフトウェア工学の基礎ワークショップ, pp.83-92, 2023年11月, 査読あり

野口 優輝, 木村 祐太, 伊原 彰則, 大平 雅雄, "自動生成されたパッチがコーディング規約違反に与える影響の分析", コンピュータソフトウェア論文誌, 査読待ち

検出された技術的負債を 説明する手法の構築(仮)

大学院での研究について

技術的負債 (Technical Debt)

ソフトウェア開発において、納期やコストを低減するために導入される質の低い設計やソースコード

- スパゲティコード
- God Classなど . .

TDが蓄積していくと . . .

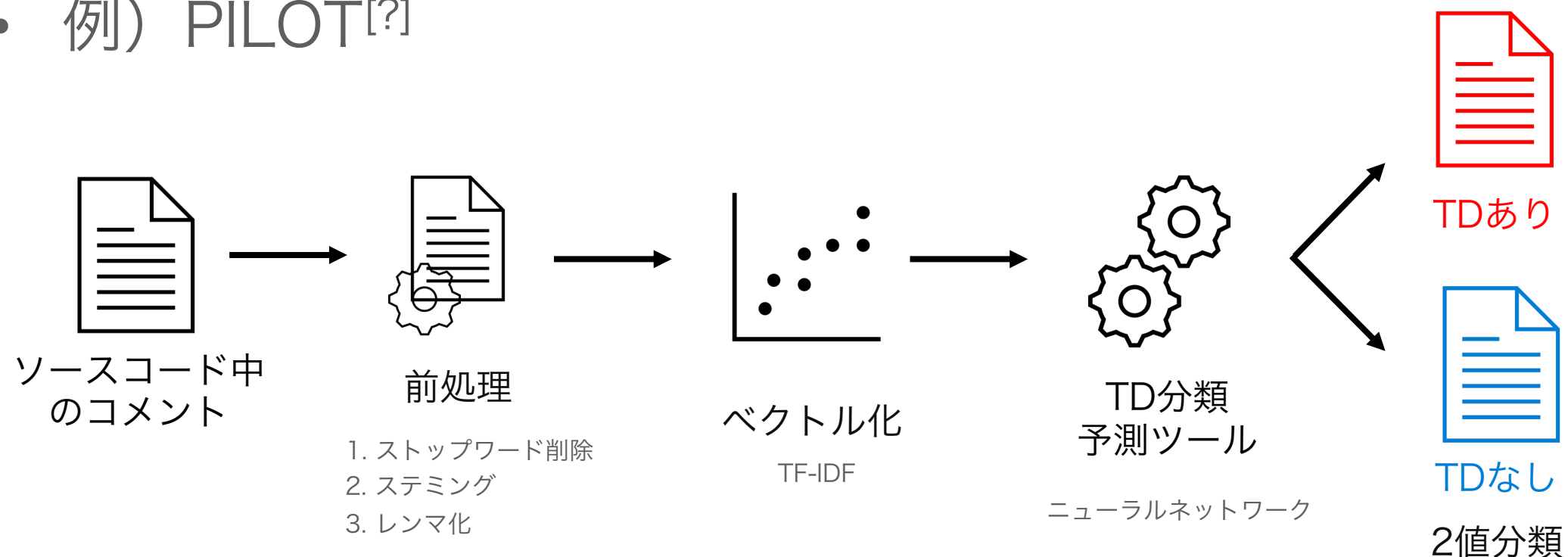
- ソフトウェアの保守性の低下^[11-12], 機能欠陥を引き起こす
→時間が経つほど, 除去するためのコストが増大

TDは早期に除去され, 適切に管理されることが望まれる

技術的負債を分類する研究

ソースコード中のコードコメントから、技術的負債が含まれているかを分類している

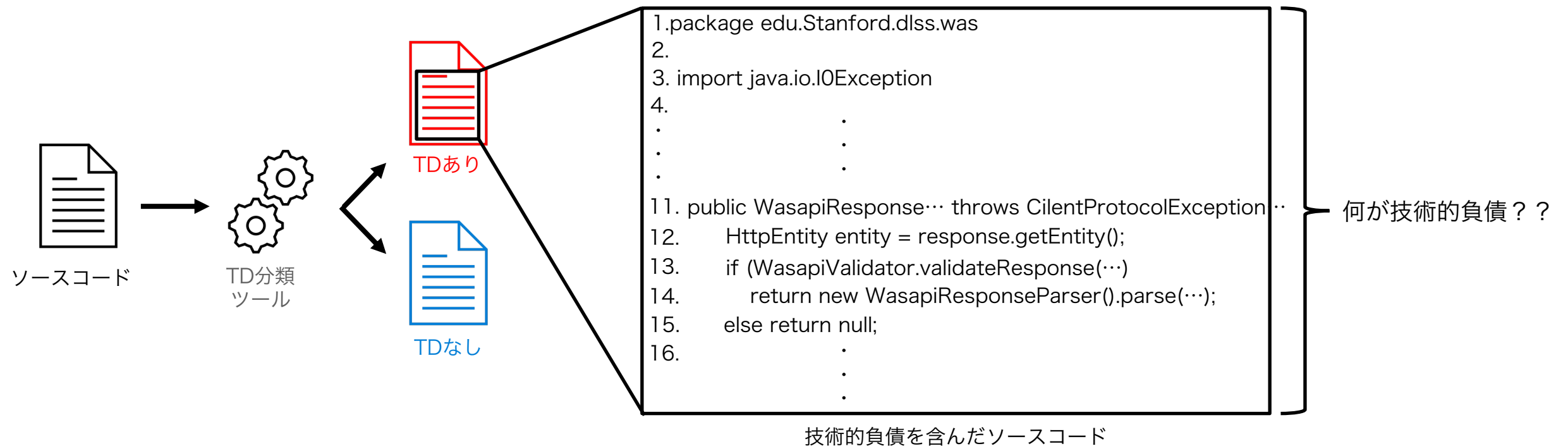
- 例) PILOT[?]



TD早期除去のために、高い精度で
技術的負債を含むファイルを分類することができた

課題点

技術的負債が含むと分類できても、ソースコード中の何が技術的負債になっているのかわからない

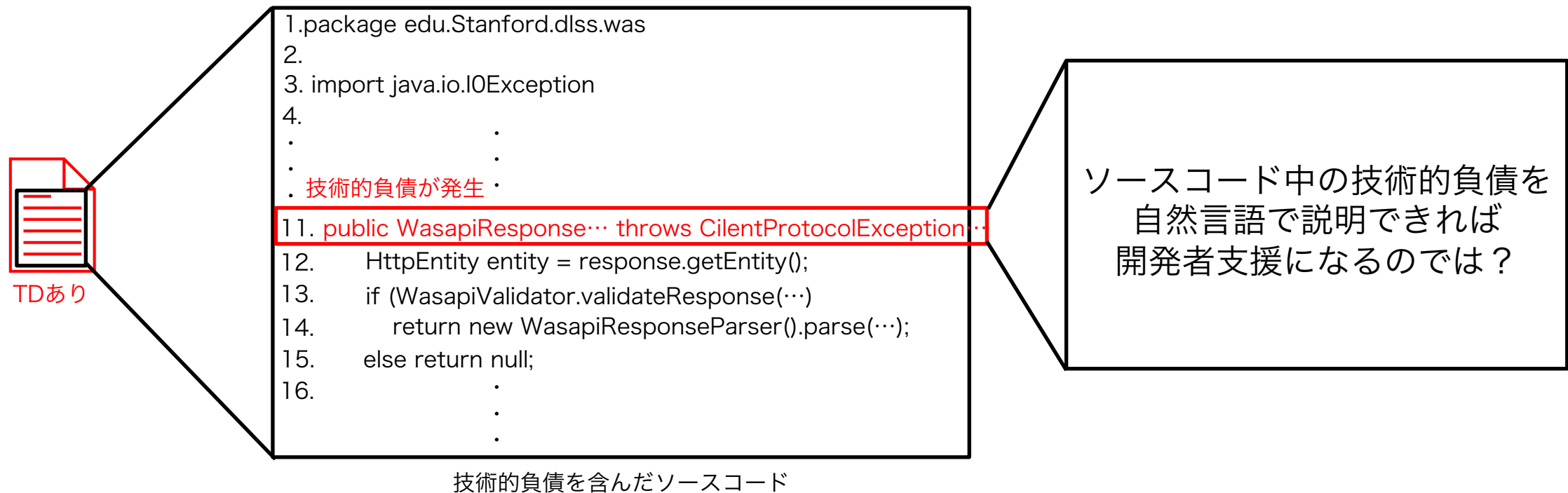


より早いTD除去のためには

開発者はソースコード中の技術的負債を理解する必要がある

本研究の動機と目的

動機



目的

- ソースコード中に含まれている技術的負債を開発者に説明するツールの構築を目指す
 - 技術的負債を含むソースコードファイルを収集
 - ソースコードから自然言語で技術的負債を要約するモデルの構築

今後の予定

技術的負債の含んだソースコードファイルの収集

アプローチの決定

- ソースコードから技術的負債を説明（自然言語）を出力する際にどのアプローチを選択するか
 - NLPモデル（Transformerなど）
 - seq2seq

こういったものを生成するか

- 「～といった技術的負債が含まれている」
 - 他にも具体的な対処法も説明するか