

開発者の観点における 技術的負債の特徴付けと 機械学習を用いた検出

オープンソースソフトウェア工学研究室

学籍番号：60219001

氏名：木村 祐太

背景

DX (Digital Transformation)

- 新たな技術の登場や顧客のニーズに合わせて容易に進化できるシステムを開発すること
- 継続的なシステムのバリューアップを可能にする

保守における課題

- 開発予算の大部分がシステム維持費として浪費^[1]
 - 日本では、開発予算の約8割がシステム維持費
- システム維持費にコストがかかりバリューアップできない

コスト増加の原因の一つとして
技術的負債への関心が高まっている

[1] 経済産業省, DXレポート ～ITシステム「2025年の崖」の克服とDXの本格的な展開～, デジタルトランスフォーメーションに向けた研究会, 2018年9月7日

技術的負債：Technical Debt [2]

時間的制約などによって導入されるソフトウェアの低品質な設計や実装

例) コードスメル：深刻な問題を抱えているコードの特徴

保守コストの増加を引き起こす[3][4][5][6]

- 開発の約23% ～ 35%が余分な作業
- 既存の技術的負債によって新たな技術的負債が導入される

開発資源（人的，金銭的，時間的）は有限

技術的負債の蓄積を抑制する必要がある

[2] Ward Cunningham. The wycash portfolio management system. SIGPLAN OOPS Mess., 4(2):29–30, 1992.

[3] Carolyn Seaman, Yuepu Guo, Nico Zazworka, Forrest Shull, Clemente Izurieta, Yuanfang Cai, and Antonio Vetr`o. Using technical debt data in decision making: Potential decision approaches. In Proceedings of the 3rd International Workshop on Managing Technical Debt, MTD, pages 45–48, 2012.

[4] Edith Tom, AybuKe Aurum, and Richard Vidgen. An exploration of technical debt. 86(6):1498–1516, 2013.

[5] Terese Besker, Antonio Martini, and Jan Bosch. The pricey bill of technical debt: When and by whom will it be paid? In Proceedings of the 33rd International Conference on Software Maintenance and Evolution, ICSME, pages 13–23, 2017.

[6] Terese Besker, Antonio Martini, and Jan Bosch. Technical debt cripples software developer productivity: A longitudinal study on developers' daily software development work. In Proceedings of the 1st International Conference on Technical Debt, TechDebt, pages 105–114, 2018.

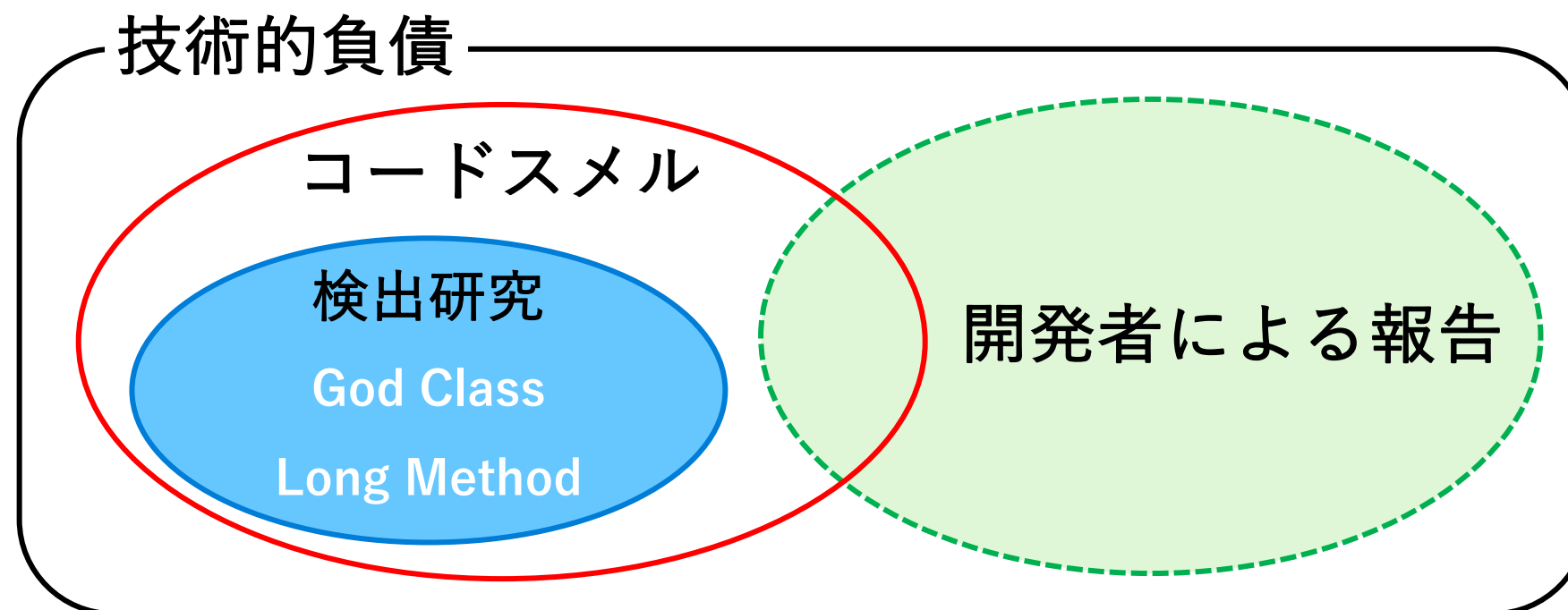
技術的負債検出に関する研究と課題

静的コード解析, 依存関係解析, 機械学習を利用^{[7][8][9]}

- コードスメルであるGod ClassやLong Methodなどを対象

課題

- 既存手法ではカバーできない**新たな技術的負債**の登場



[7] Nico Zazworka, Antonio Vetr`o, Clemente Izurieta, Sunny Wong, Yuanfang Cai, Carolyn Seaman, and Forrest Shull. Comparing four Approaches for Technical Debt Identification. Software Quality Journal, Vol. 22, pages 1–24, 2013.

[8] John Brondum and Liming Zhu. Visualising Architectural Dependencies. In Proceedings of the 3rd International Workshop on Managing Technical Debt, MTD, pages 7–14, 2012.

[9] Daniel Cruz, Amanda Santana, and Eduardo Figueiredo. Detecting Bad Smells with Machine Learning Algorithms: An Empirical Study. In Proceedings of the 3rd International Conference on Technical Debt, TechDebt, pages 31–40, 2020.

課題管理システムを用いた報告

TD Issue^[10]

- 課題票（Issue）にTDラベルを割り当てて明示される事象

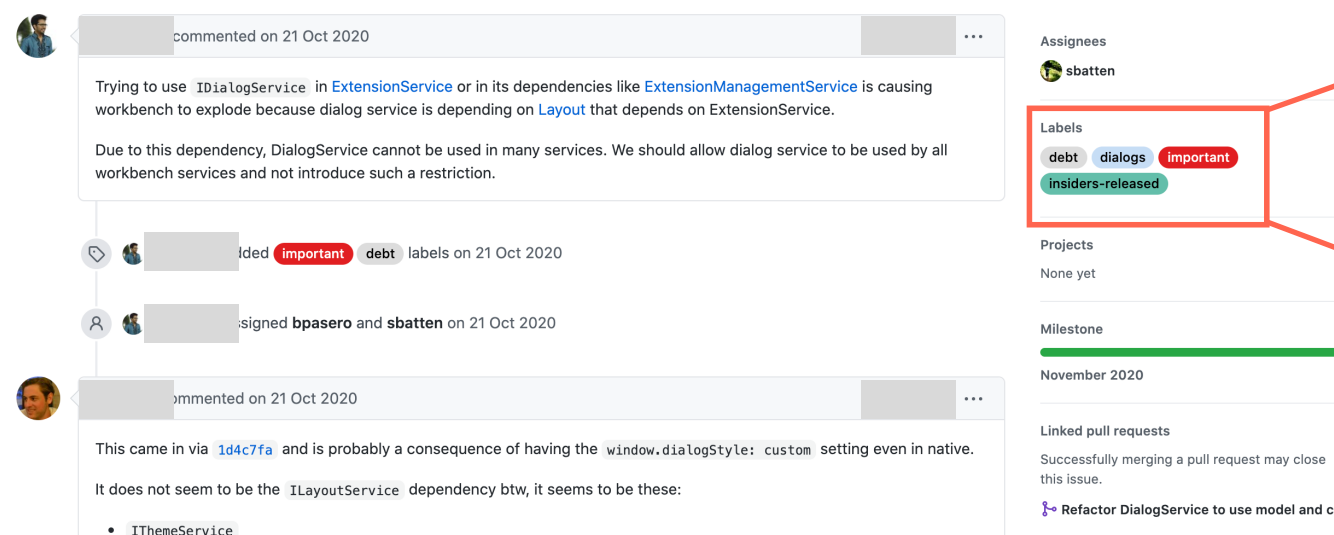
Issue

課題管理システムに記録される報告のこと

不具合，機能改善，機能要求などさまざまな課題が管理される

Cannot use IDialogService in some services due to custom dialog service dependencies
#109056

 Closed sandy081 opened this issue on 21 Oct 2020 · 13 comments



“debt”というTDラベルが
割り当てられている

引用：<https://github.com/microsoft/vscode/issues/109056>

[10] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. Beyond the code: Mining self-admitted technical debt in issue tracker systems. In Proceedings of the 17th International Conference on Mining Software Repositories, MSR, pages 137–146, 2020.

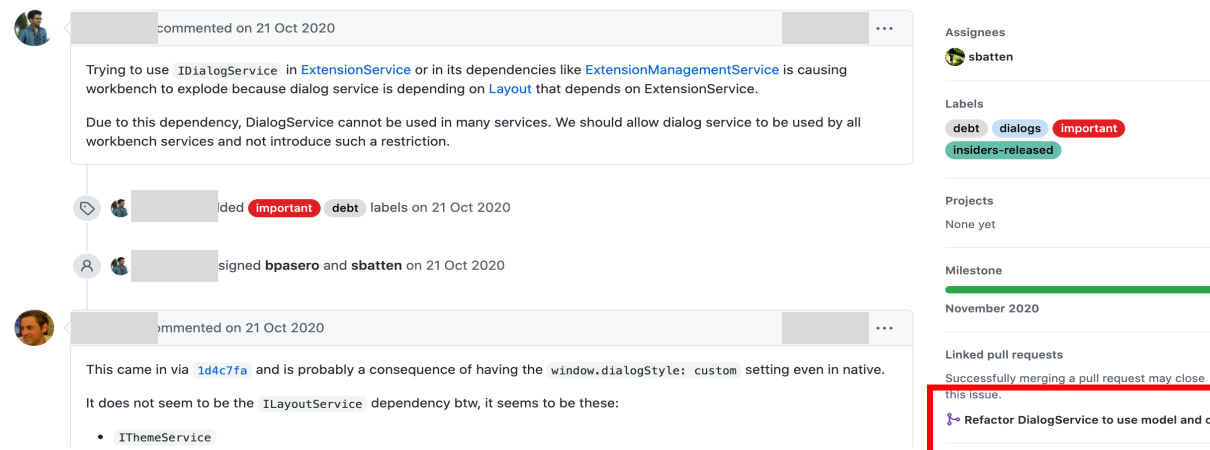
着眼点と目的

着眼点

- TD-Issueを分析することで、開発者が扱う技術的負債の特徴が理解できる
 - 変更情報から技術的負債が含まれているファイルなどが特定できる

Cannot use IDialogService in some services due to custom dialog service dependencies
#109056

 Closed sandy081 opened this issue on 21 Oct 2020 · 13 comments



commented on 21 Oct 2020

Trying to use `IDialogService` in `ExtensionService` or in its dependencies like `ExtensionManagementService` is causing workbench to explode because dialog service is depending on `Layout` that depends on `ExtensionService`.

Due to this dependency, `DialogService` cannot be used in many services. We should allow dialog service to be used by all workbench services and not introduce such a restriction.

added **important** **debt** labels on 21 Oct 2020

signed **bpassen** and **sbatten** on 21 Oct 2020

commented on 21 Oct 2020

This came in via [1d4c7fa](#) and is probably a consequence of having the `window.dialogStyle: custom` setting even in native. It does not seem to be the `ILayoutService` dependency btw, it seems to be these:

- `IThemeService`

Assignees
sbatten

Labels
debt **dialogs** **important** **insiders-released**

Projects
None yet

Milestone
November 2020

Linked pull requests
Successfully merging a pull request may close this issue.
[Refactor DialogService to use model and c...](#)

この課題に対する変更情報が紐付けられている

目的

- 開発者が扱う技術的負債の検出を支援する
 - ソースコードの構造的特徴や意味的特徴を利用した検出手法の構築

研究の取り組み

(P1) 技術的負債に関連する課題票の分析と分類

- TD-Issueの特徴分析と自動分類

(P2) 課題票で扱われる技術的負債の分析と分類

- TD-Issueの修正対象ファイルの分析と自動分類

(P3) 深層学習を用いた技術的負債の検出

- ソースコードそのものを入力とした検出

概要

課題票で扱われる技術的負債の分析と分類

- ソースコードメトリクスを用いて技術的負債を理解する
- 決定木アルゴリズムを用いてパターンを分析する
 - パターンごとに技術的負債を確率的に表現

研究意義

- 開発者が技術的負債を定量的に判断することができる
 - ソースコードメトリクスの計測のみで可能

対象プロジェクト

470プロジェクトのリポジトリを対象に分析を実施

- Java：151リポジトリ，Python：319リポジトリ
 - 既存研究のデータセット^[11]をフィルタリングし利用

フィルタリング

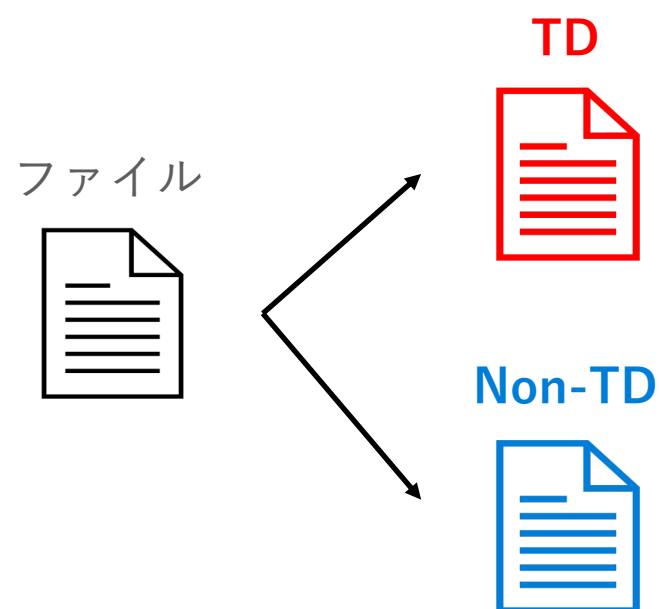
- 公開されているリポジトリであること
- 過去に技術的負債の除去が行われていること

[11] D. Skryseth, K. Shivashankar, I. Pilán and A. Martini, "Technical Debt Classification in Issue Trackers using Natural Language Processing based on Transformers," In Proceedings of 6th International Conference on Technical Debt (TechDebt), pp. 92-101, 2023

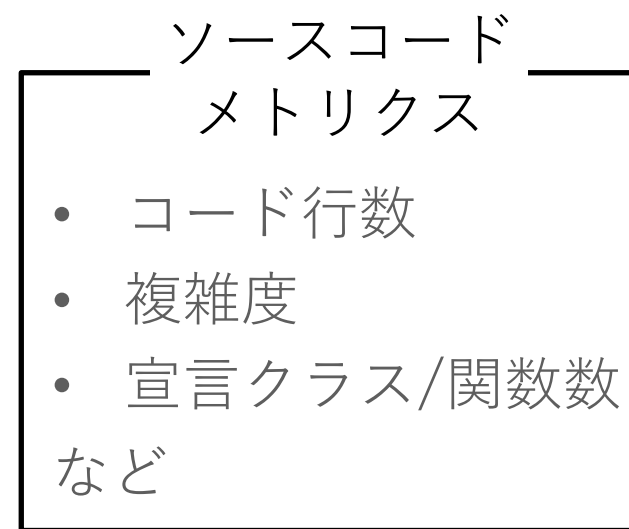
分析手順

1. 技術的負債を含むファイル（TDファイル）の特定
2. ソースコードメトリクス計測
3. 決定木アルゴリズムを用いたパターン分析

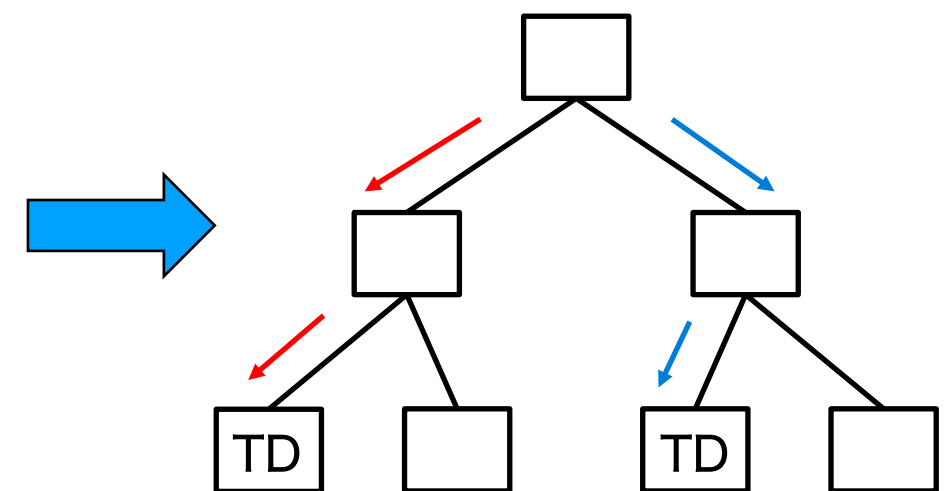
1. ファイルの特定



2. メトリクスの計測



3. 決定木の構築と分析



分析手順 | TDファイルの特定

ファイルの定義

TDファイル

- TD-Issueの解決時に対象となったバージョンのファイル

Non-TDファイル

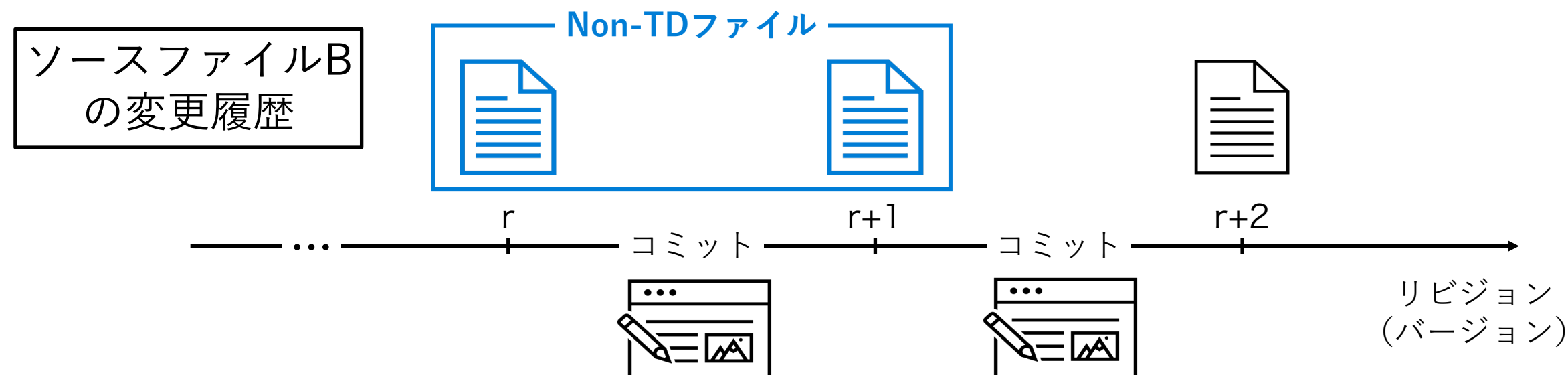
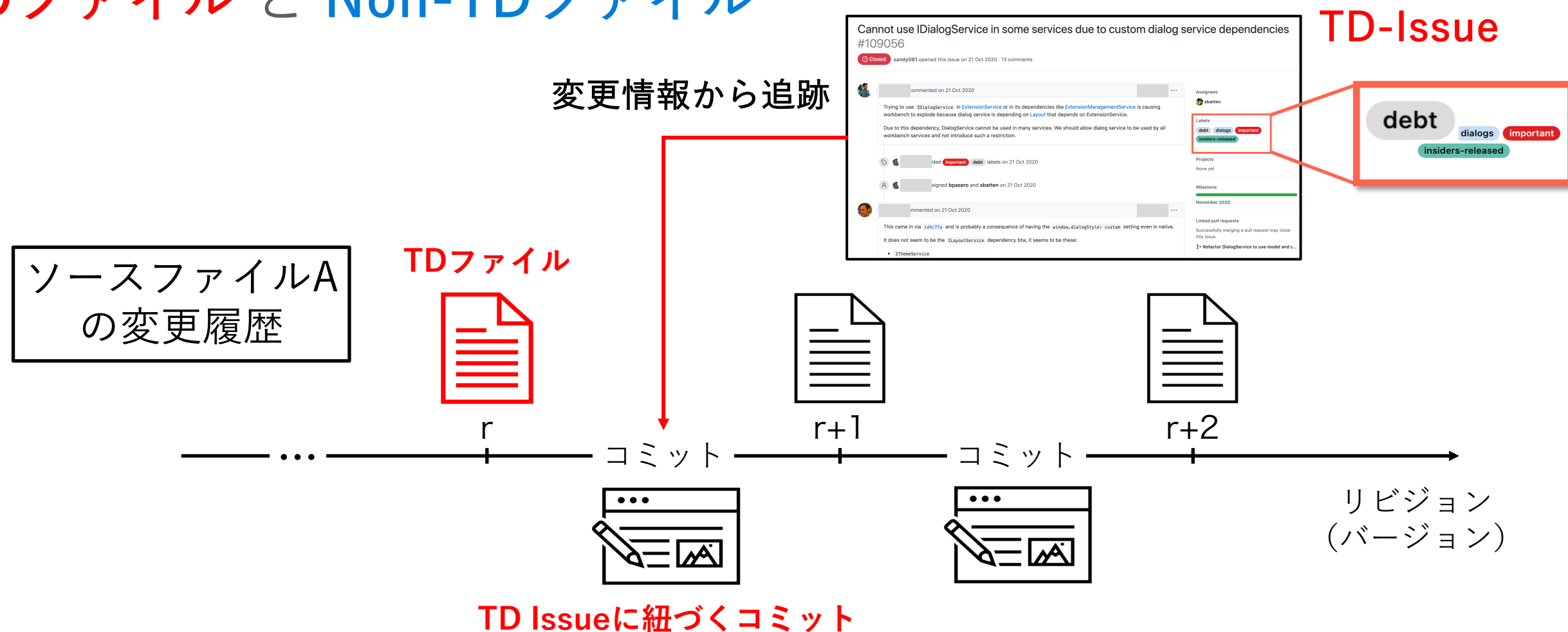
- 一度もTD-Issueの解決に関与したことがない全バージョンのファイル

フィルタリング

- ソースファイル以外のファイルを除外
 - テストファイル, ドキュメントなど

P2：課題票で扱われる技術的負債の分析と分類

TDファイルと Non-TDファイル



分析手順 | メトリクス計測

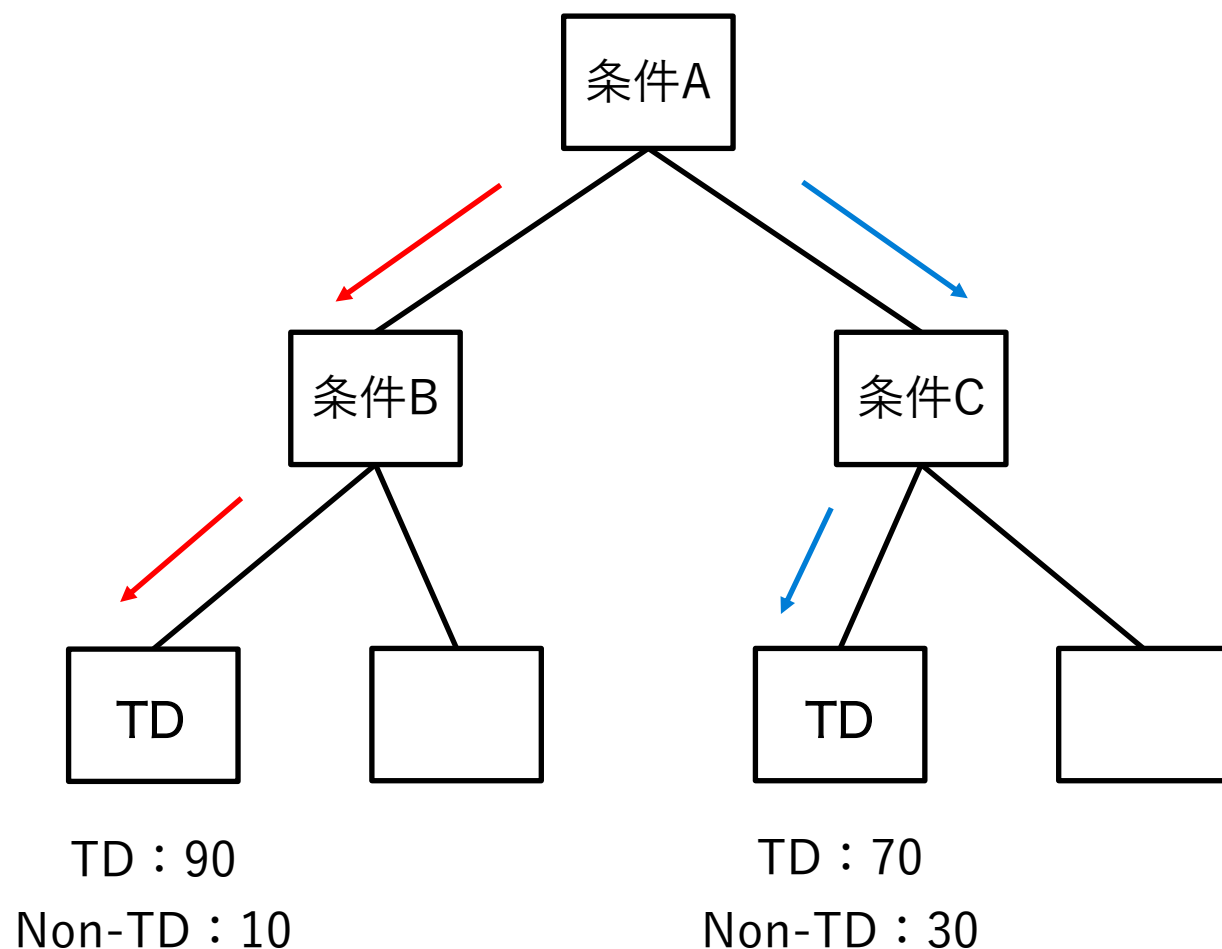
ファイル内の状態を表すために、以下のメトリクスを利用

メトリクス	例	メトリクス数
行数	<u>コード行数</u> 空白行やコメントを除いた コードの行数	5種類
複雑度	<u>サイクロマティック複雑度</u> 条件分岐など実行経路の複雑さ	7種類
宣言数	<u>宣言クラス数</u> ファイルで定義されるクラス数	10種類

分析手順 | パターン分析

決定木アルゴリズムで学習し，TDと判断される確率が高いパスの条件を分析する

- 反対に，Non-TDと判断される確率が高いパスも分析



パターン1

→ (条件A) ∩ (条件B)

$$90 / (90 + 10) = 0.9$$

パターン2

→ (条件A) ∩ (条件C)

$$70 / (70 + 30) = 0.7$$

データセット

TD Issueと課題管理システムから追加収集したTD Pull Requestの件数

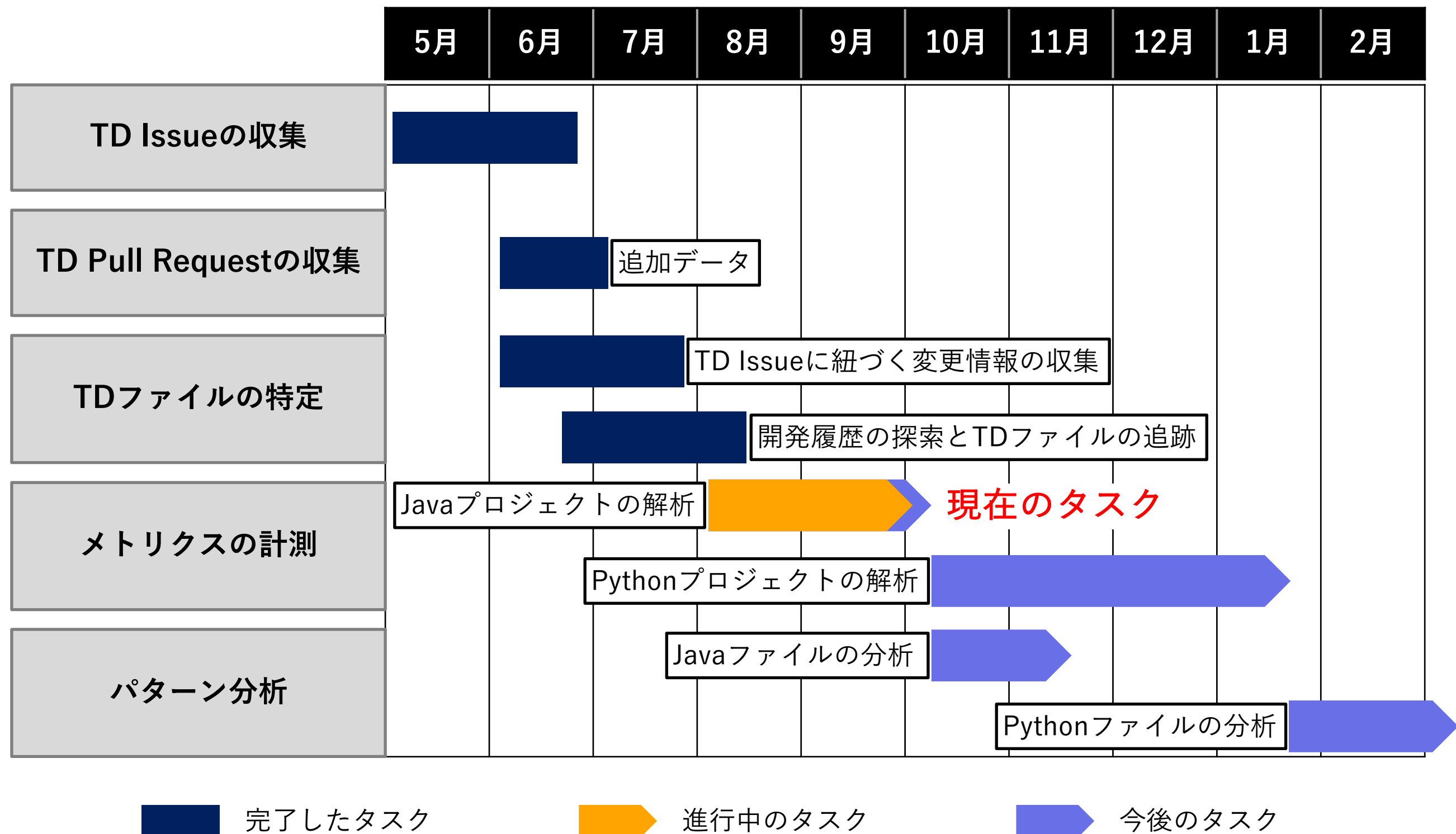
	リポジトリ数	TD Issue	TD Pull Request
Java	151	3,756	1,473
Python	319	6,630	2,018

追加データ分

TDファイルとNon-TDファイルの件数

	リポジトリ数	TDファイル	Non-TDファイル
Java	151	35,758	1,413,606
Python	319	24,283	1,011,726

現在の進捗



データセットの分析

技術的負債はファイルの更新に影響があるか？

- 開発履歴からTDファイルとNon-TDファイルの更新間隔を計算

$$\text{更新間隔 (日 / 回)} = \frac{\text{最終更新日時} - \text{追加日時}}{\text{変更回数}}$$

技術的負債が発生しやすいコンポーネントは？

- コンポーネントごとにリポジトリの割合を計算
 - コンポーネントは役割ごとに配置されていると仮定

ファイルパス：AAA / analyse / a.java

↑
コンポーネント

analyse

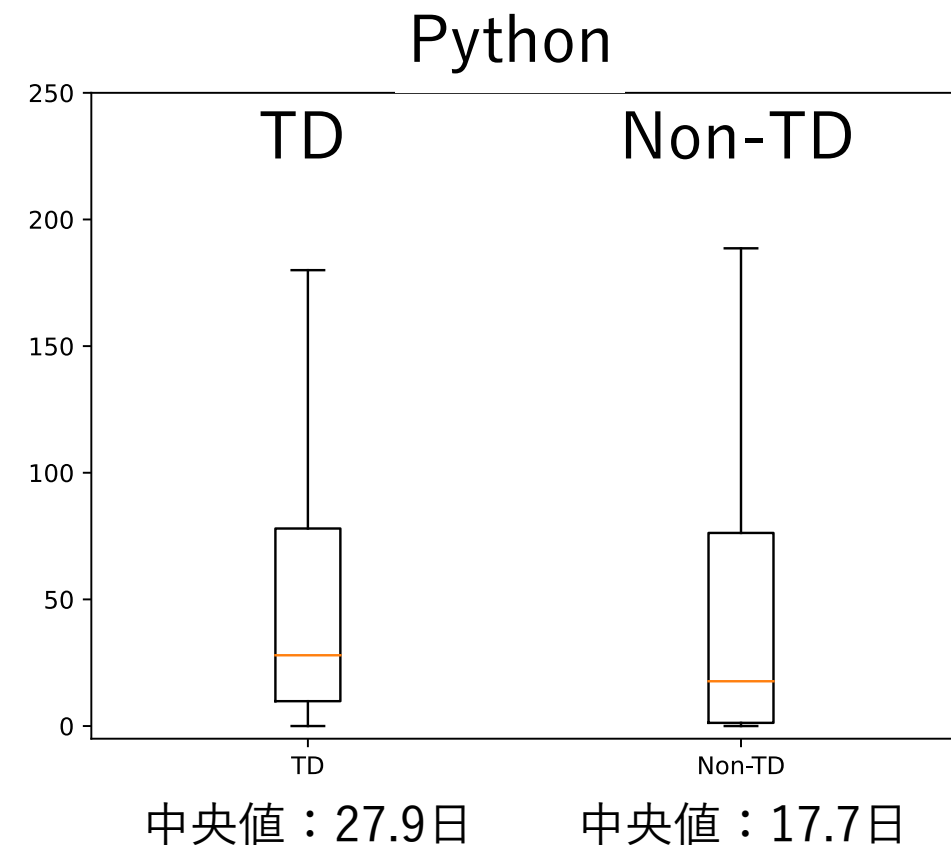
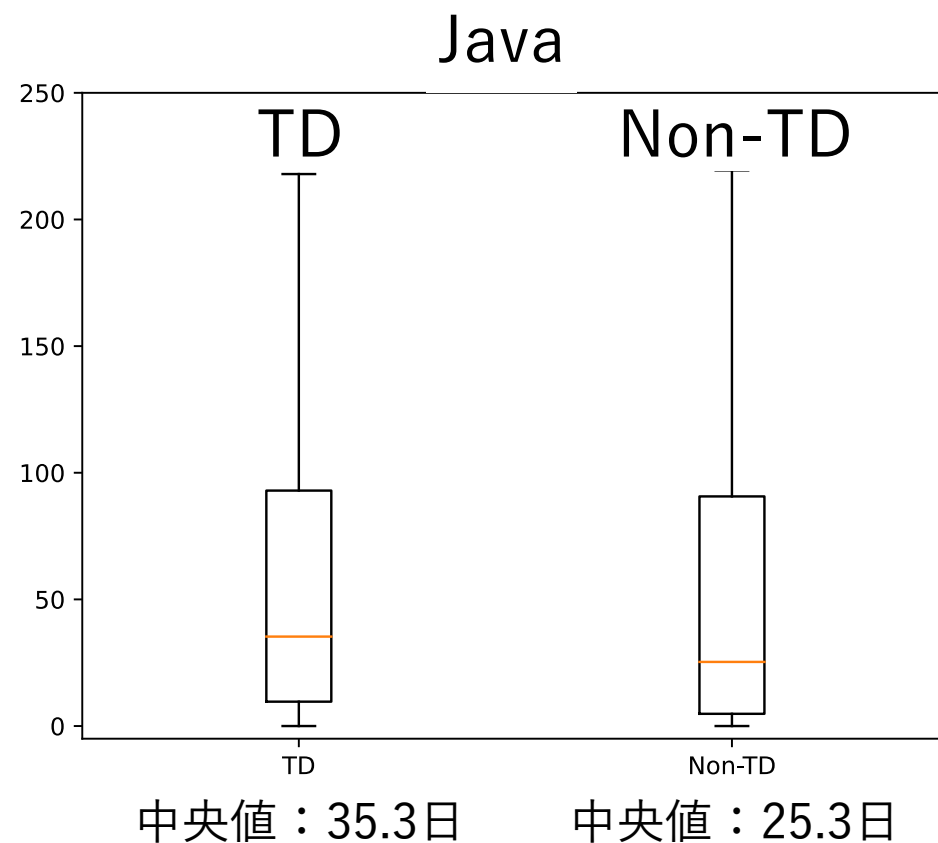
TDが発生した
リポジトリ数

3 ÷ 10 = 0.3

ファイルの更新間隔

TDファイルはNon-TDファイルと比べて、約10日ほど更新が遅い

- TD Issueは、そのほかのIssueと比べて、変更量が多く解決にかかる時間が長い^[12]ことが関係している

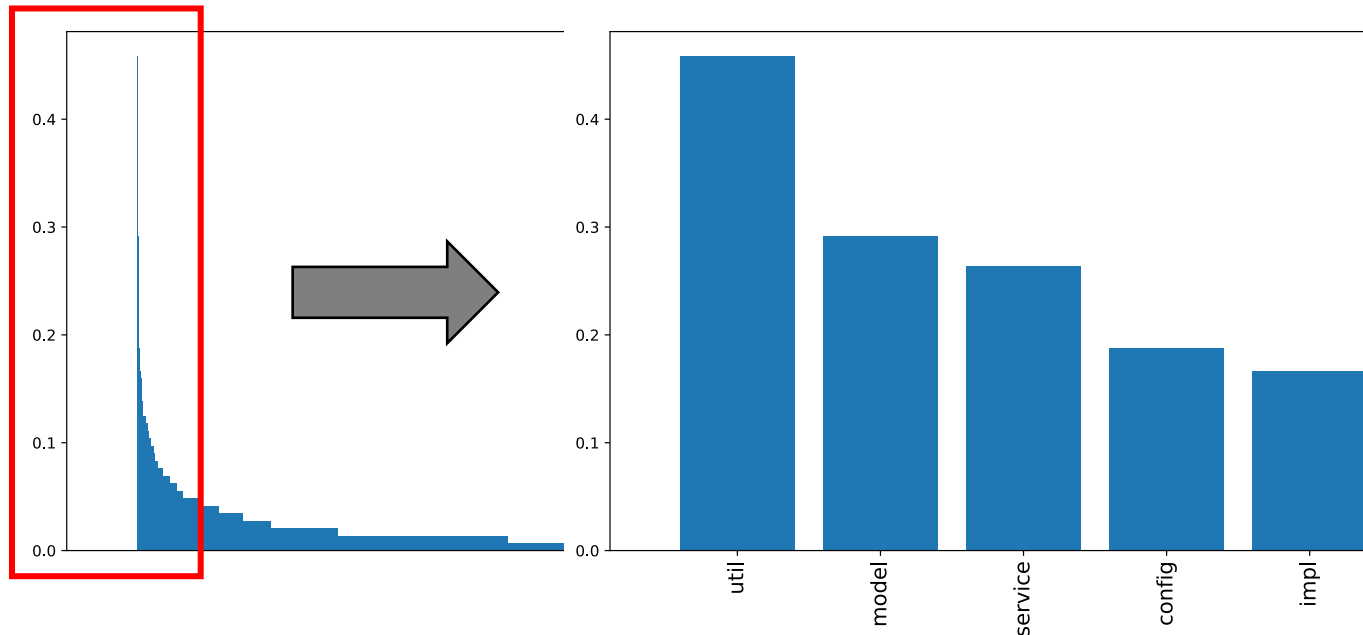


技術的負債は開発の妨げになっている

[12] 木村 祐太, 大平 雅雄, “技術的負債に関連する課題票分類手法の構築”, 情報処理学会, 2023, 64巻, 1月号

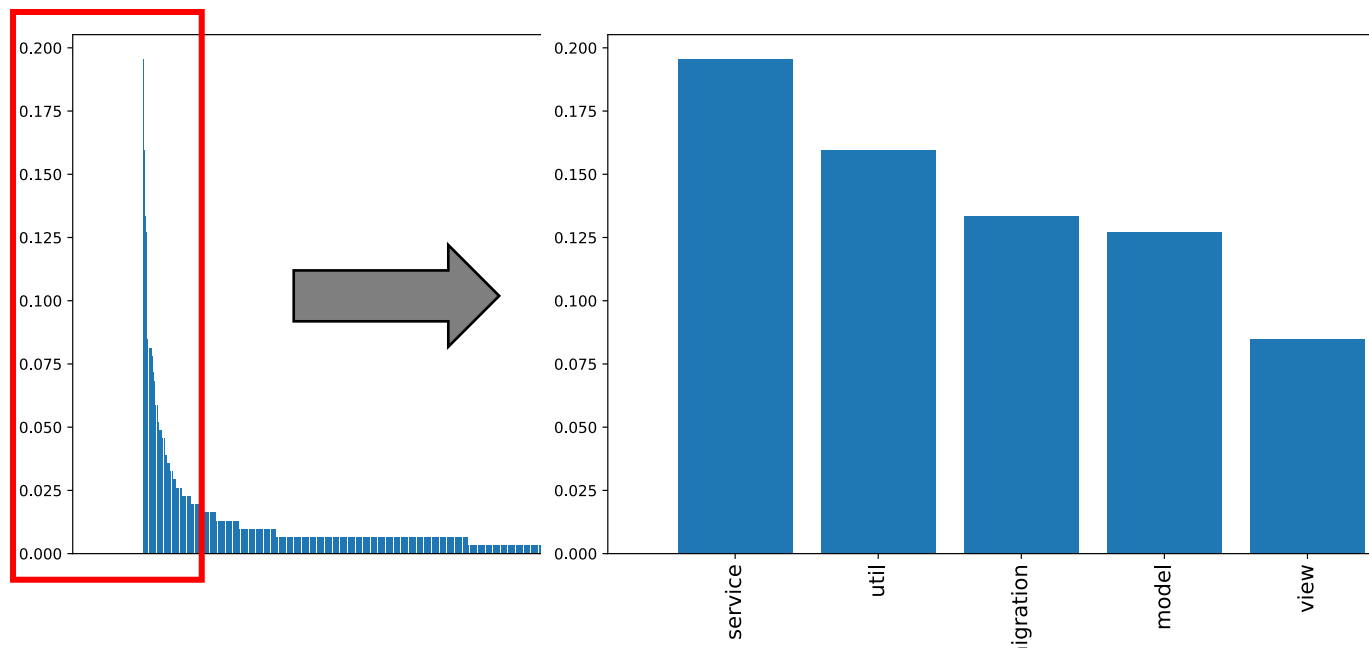
技術的負債とコンポーネント

Javaプロジェクト



コンポーネント	リポジトリ
util	0.46
model	0.29
service	0.26
config	0.19
impl	0.17

Pythonプロジェクト



コンポーネント	リポジトリ
service	0.20
util	0.16
migration	0.13
model	0.13
view	0.08

他のファイルで呼び出されやすいコンポーネントで発生しやすい？

まとめ

分析のまとめ

- TDファイルは更新間隔が長い
 - 技術的負債を早期に検知することの有用性を示唆
- 技術的負債が発生しやすいコンポーネントの存在
 - 技術的負債の検出のヒントになる可能性

今後の予定

- メトリクスの計測が終わり次第、分析を実施
- ほかに検出のヒントになる観点を検討