

静的コード解析ツールが検出する技術的負債と 課題票が対象とする技術的負債の特徴比較: 技術的負債の優先順位付けに向けた調査

オープンソースソフトウェア工学研究室
学籍番号：60230113
氏名：松尾 和輝

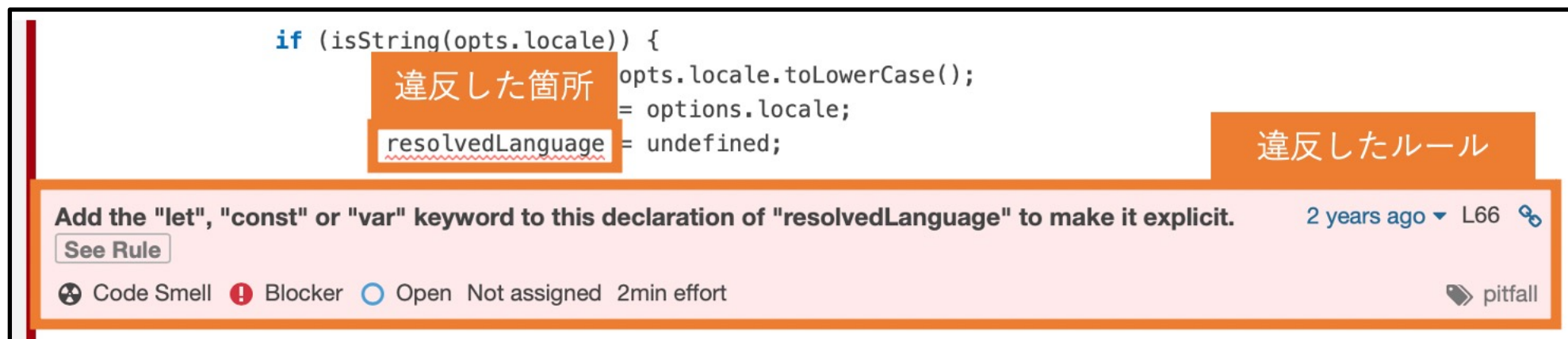
技術的負債（Technical Debt, TD） [1]

- 短期的な利益を求めて最適でない実装を行なったために、後々修正が必要となるコードや設計のこと
- 静的コード解析ツールによって検出されるTD（TD-Tool）や、SATD-Code, SATD-Issueなどがある
- TDはプログラムの品質の低下や保守コストの増加などを引き起こし、ソフトウェアに悪影響を及ぼすため、早期に返済する必要がある

[1] Ward Cunningham. The wycash portfolio management system. In 7th Objectoriented Programming Systems, Languages, and Applications (OOPSLA), pp. 29–30,1992.

静的コード解析ツール

- コードを実行せずにコードの脆弱性, バグ, TDなどを検出できるツール
- ルールに基づいて自動でTDを検出するため, 開発者がTDを探す手間を減らすことができる



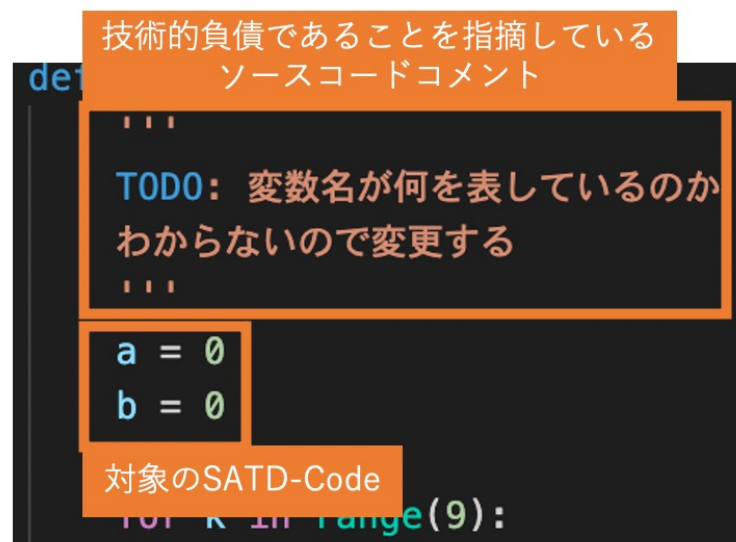
検出されたTD-Tool

- 本研究ではSonarQube¹を利用する

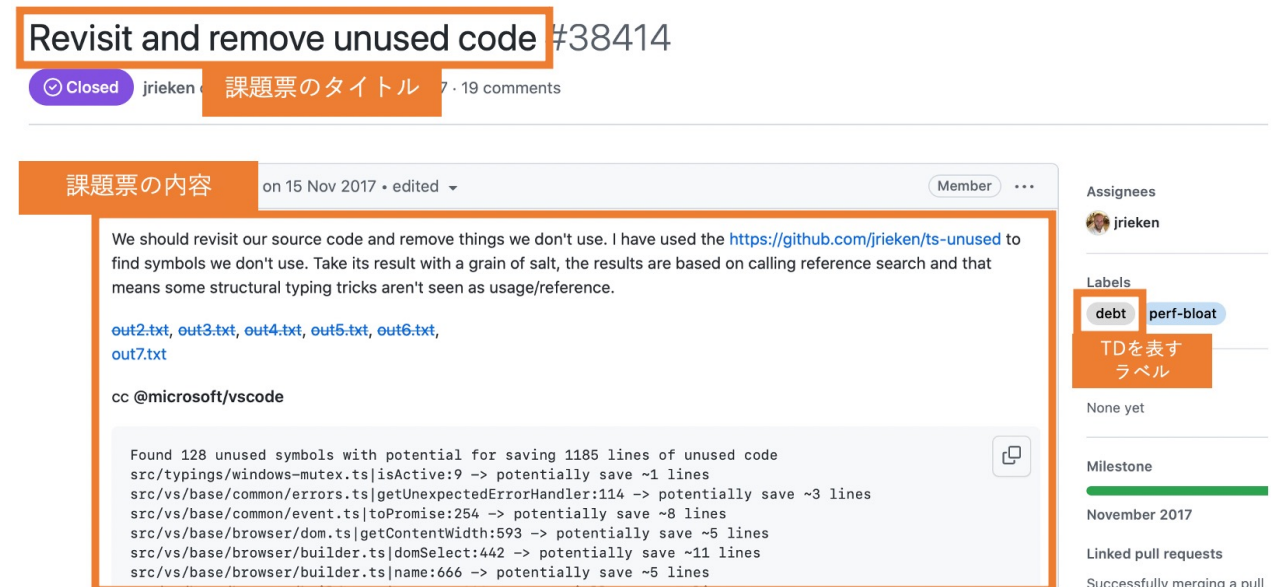
¹ <https://www.sonarqube.org/>

Self-Admitted TD (SATD)

- 開発者がTDであることを文書化し明示するTD
 - SATD-Code^[2]
 - ソースコードコメントを用いて明示しているSATD
 - SATD-Issue^[3]
 - 課題票にTDを表すラベルを付与することで明示しているSATD



SATD-Codeを指摘しているソースコードコメントと、指摘されたSATD-Code



SATD-Issueを指摘している課題票

[2] Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In 30th International Conference on Software Maintenance and Evolution (ICSME), pp. 91–100, 2014.

[3] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. Beyond the code: Mining self-admitted technical debt in issue tracker systems. MSR '20: Proceedings of the 17th International Conference on Mining Software Repositories, pp. 137–146, 2020.

背景

- プロダクトの保守性を高めるには、継続的に混入されるTDを返済する必要がある
- 開発には時間、人数などのリソースに限りがあり、全てのTDを返済できない
- TDに優先順位をつけることで、効率的にTDを返済する必要がある
- 現状ではTD-ToolやSATD-Code, SATD-Issueの3種類のTDを考慮した優先順位付けの手法は存在しない

目的とアプローチ

- 目的
 - 優先順位付けにおける知見を得るために，TD-ToolとSATD-Issueの特徴の違いを明らかにする
 - 本研究では，TD-ToolとSATD-Issueのみに着目する
- アプローチ
 - TD-ToolとSATD-Issueの特徴を比較する

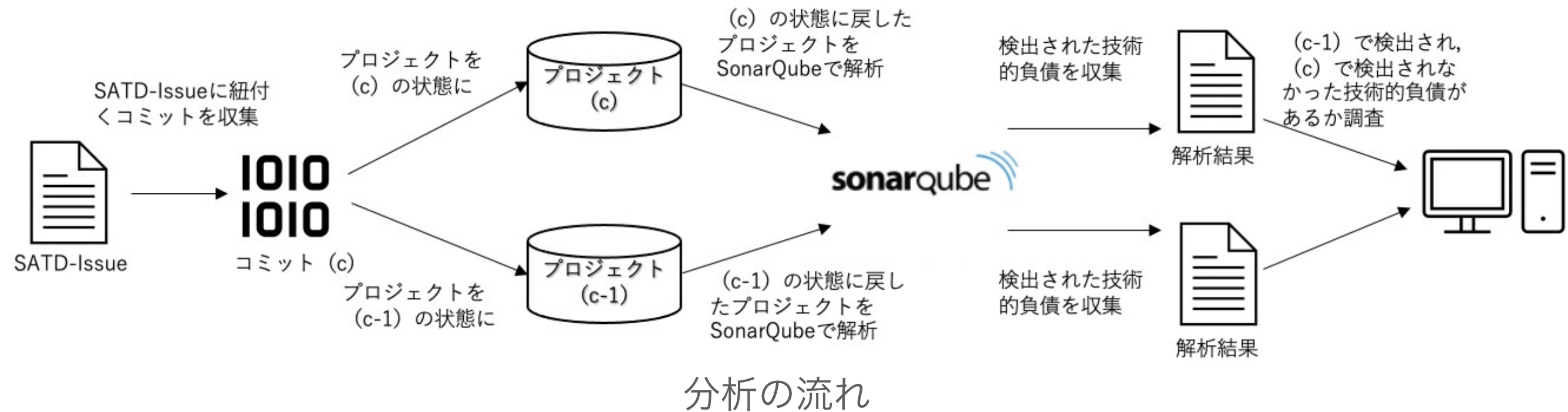
リサーチクエスチョン

- RQ1 : SATD-IssueとTD-Toolで共通するTDとはどのようなものか
 - SATD-Issueと紐付けられるTD-Toolを抽出
 - SATD-IssueとTD-Toolをカテゴリーごとに分類
 - 共通するTDの特徴を分析
- RQ2 : SATD-IssueとTD-Toolにどのような違いがあるか
 - 技術的負債の返済期間と, 返済に必要なコードの変更行数でSATD-IssueとTD-Toolに違いがあるかを分析
 - SATD-IssueとTD-Toolを分類し, マンホイットニーのU検定を行う

分析手順：RQ1

RQ1：SATD-IssueとTD-Toolで共通するTDとはどのようなものか？

- 対象プロジェクトはVSCoDeとする



データセット：RQ1, RQ2

- 846件のSATD-Issueと14,252件のTD-Toolをデータセットとする
- SATD-IssueとTD-Toolをそれぞれ分類する
 - SATD-Issue
 - 以下の13種類に分類する

UI	Build	Code Style	Security	Requirement	Architecture	Code Duplication
Test	Infrastructure	Performance	Document	Complex Code	Clean Up	

– TD-Tool

- 以下の15種類に分類する（数が少ないルールは除外）

Cases should end with break	Remove unused variable	Not use comma	Remove extra semicolon	Not shadowed variable	Not redundant boolean	Switch have at least 3 case
Variables declared before used	Remove dead stores	Not have same implementation	Not always evaluate true or false	Not redeclared variable	Not have identical implementations	Avoid redundant cast and not-null assertion
Not duplicate selector						

結果：RQ1

- SATD-IssueとTD-Toolで共通するTDとはどのようなものか？

調査結果

TD-Toolを扱っている SATD-Issueの件数	77 (9.1%)
TD-Toolを扱っていない SATD-Issueの件数	769 (90.9%)

- SATD-Issueの内、9.1%がTD-Toolと紐付けられる
(共通するTDを扱う)
 - ほとんどのSATD-IssueはTD-Toolと紐付けできない

結果：RQ1

- Code Style, Buildなどに分類されるSATD-Issueは1件もTD-Toolと共通しておらず, Requirement, Clean Upに分類されるSATD-IssueなどはTD-Toolと共通しているものが多いという違いが見られた

SATD-Issueの分類結果

カテゴリー	TD-Toolを扱っている SATD-Issue の件数	TD-Toolを扱っている SATD-Issue の割合	SATD-Issue の分類結果
UI	3	4.8%	63
Test	4	7.0%	57
Performance	4	9.3%	43
Infrastructure	2	7.1%	28
Document	0	0.0%	10
Code Style	0	0.0%	13
Build	0	0.0%	12
Security	0	0.0%	0
Requirement	3	15.0%	20
Complex Code	17	7.3%	232
Architecture	30	11.3%	266
Clean Up	13	15.3%	85
Code Duplication	1	5.9%	17
合計	77		846

分析手順：RQ2

RQ2：SATD-IssueとTD-Toolにどのような違いがあるか

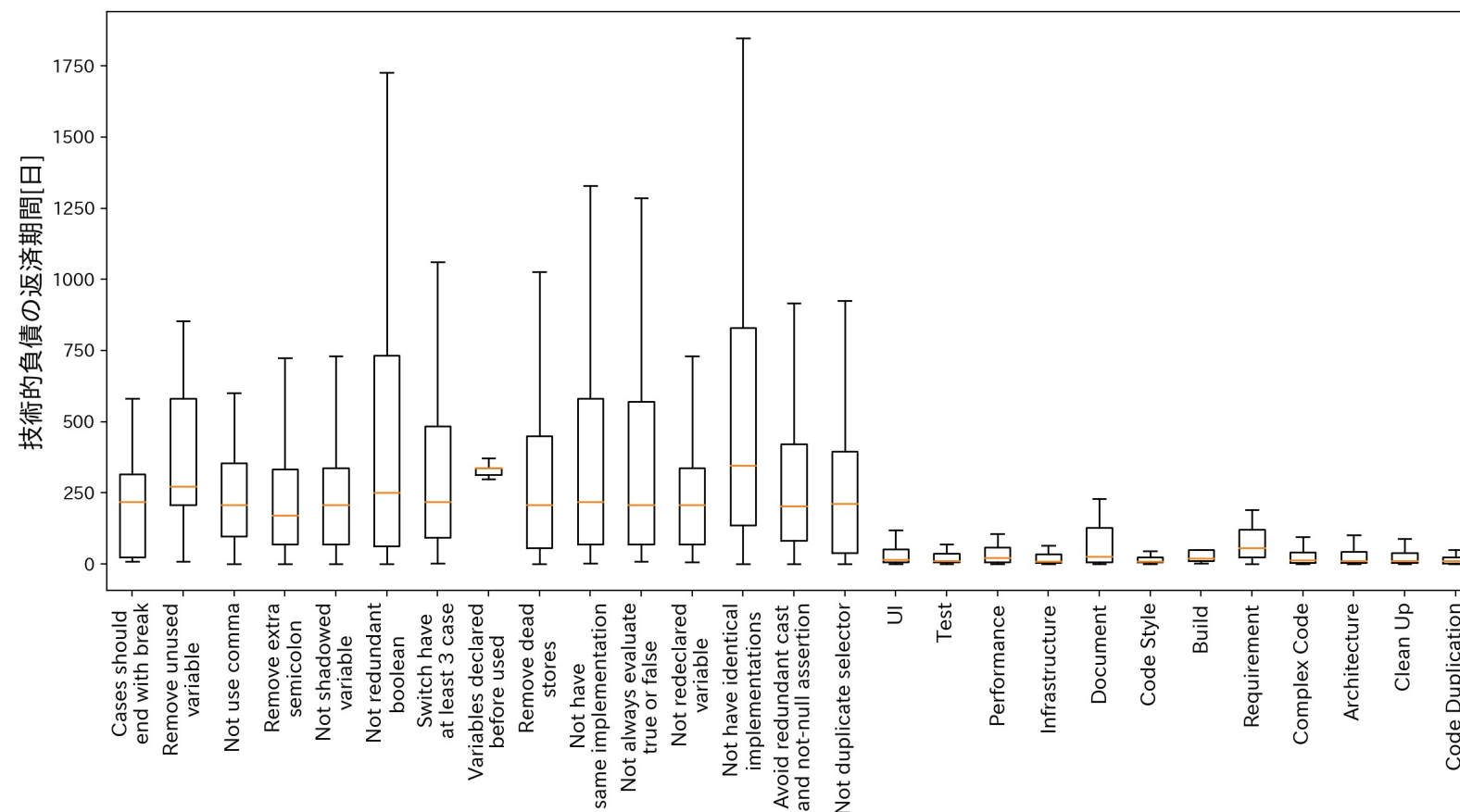
- 技術的負債の返済期間
 - どのようなTDが返済に時間がかかるのかを明らかにするため
 - TDの混入から返済までの期間を算出
- 返済に必要なコードの変更行数
 - どのようなTDが返済に労力がかかるのかを明らかにするため
 - TDを返済しているコミットが加えたコードの追加行と削除行の合計を算出

結果：RQ2

RQ2：SATD-IssueとTD-Toolにどのような違いがあるか

- 技術的負債の返済期間
 - 全体的にTD-Toolの方がSATD-Issueよりも返済期間が長いという違いが見られた

TD-ToolとSATD-Issueの返済期間に関する箱ひげ図

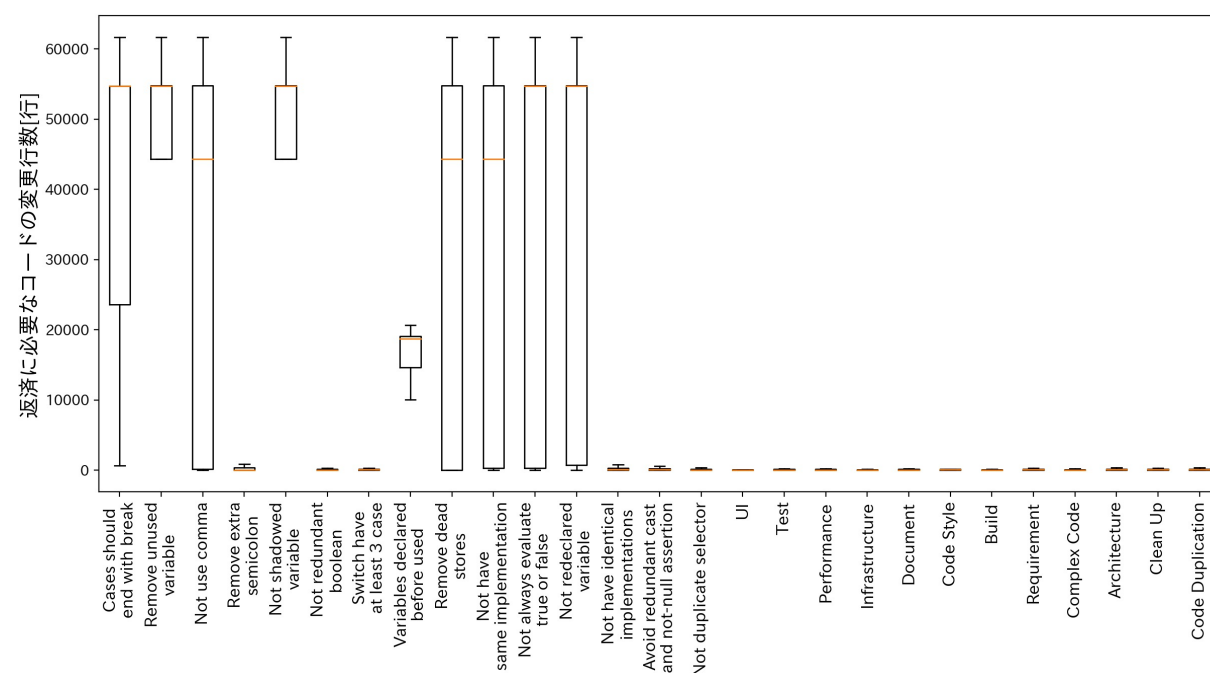


結果：RQ2

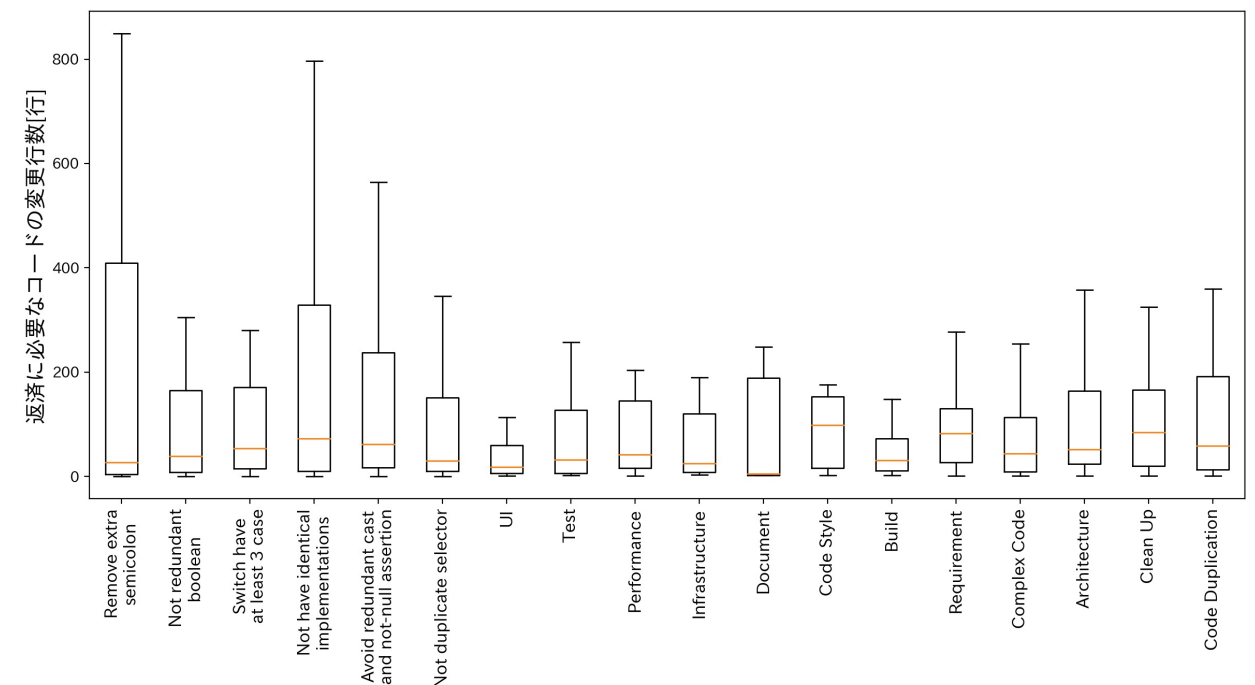
RQ2：SATD-IssueとTD-Toolにどのような違いがあるか

- 返済に必要なコードの変更行数
 - TD-Toolの多くは返済に必要なコードの変更行数が多いが、一部のTD-ToolはSATD-Issueよりも返済に必要なコードの変更行数が少ないという違いが見られた

全体（左図）



抜粋（右図）



考察：RQ1

- SATD-IssueとTD-ToolでTDが共通していることから，開発者と静的コード解析ツールの両方がそのTDを問題視していると考えられる
- そのため，Architecture, Requirement, Clean Upに分類されるSATD-Issueから早期に返済すべきと考えられる

考察：RQ2

- 多くのTD-Toolは返済期間が長く返済に必要なコードの変更行数も多いことから、返済が困難であると考えられるため、早期に返済すべきと考えられる
- 一部のTD-Toolは返済期間が長いが返済に必要なコードの変更行数は少ない傾向にあり、放置されていると考えられるため、後回しにしても良いと考えられる
- SATD-Issueは返済期間が短く返済に必要なコードの変更行数も比較的少ない傾向にあり、返済しやすいと考えられるため、後回しにしても良いと考えられる

まとめ

- 目的
 - 優先順位付けにおける知見を得るために、TD-ToolとSATD-Issueの特徴の違いを明らかにする
- アプローチ
 - TD-ToolとSATD-Issueの特徴を比較する
- 結果
 - SATD-Issueの内、9.1%がTD-Toolと共通するTDを扱う
 - 多くのTD-Toolは寿命が長く返済に必要なコードの変更行数も多い
 - 一部のTD-Toolは寿命が長いが返済に必要なコードの変更行数は短い
 - SATD-Issueは寿命が短く返済に必要なコードの変更行数も比較的少ない

大学院での研究について

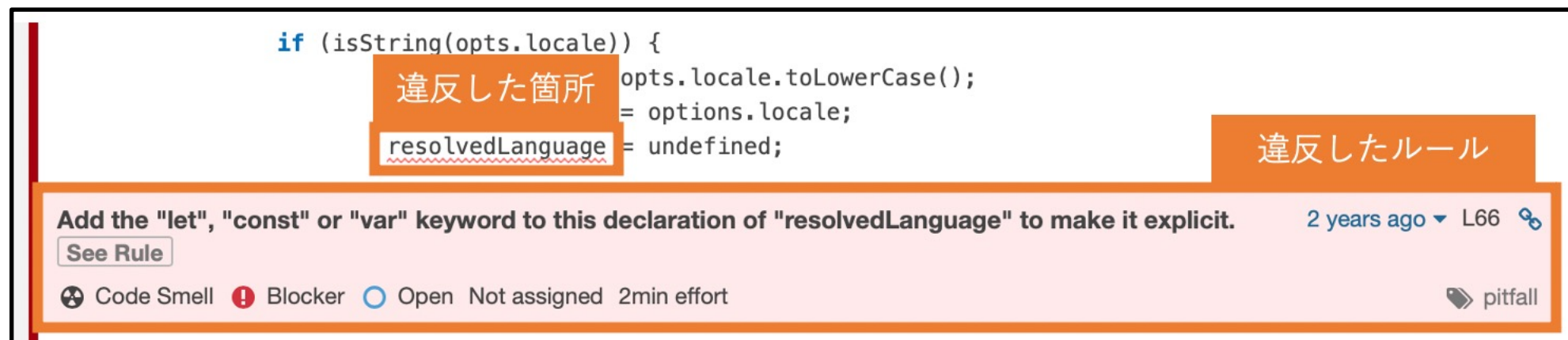
技術的負債（Technical Debt, TD） [1]

- 短期的な利益を求めて最適でない実装を行なったために、後々修正が必要となるコードや設計のこと
- 静的コード解析ツールによって検出されるTD（TD-Tool）や、SATD-Code, SATD-Issueなどがある
- TDはプログラムの品質の低下や保守コストの増加などを引き起こし、ソフトウェアに悪影響を及ぼすため、早期に返済する必要がある

[1] Ward Cunningham. The wycash portfolio management system. In 7th Objectoriented Programming Systems, Languages, and Applications (OOPSLA), pp. 29–30,1992.

静的コード解析ツール

- コードを実行せずにコードの脆弱性, バグ, TDなどを検出できるツール
- ルールに基づいて自動でTDを検出するため, 開発者がTDを探す手間を減らすことができる



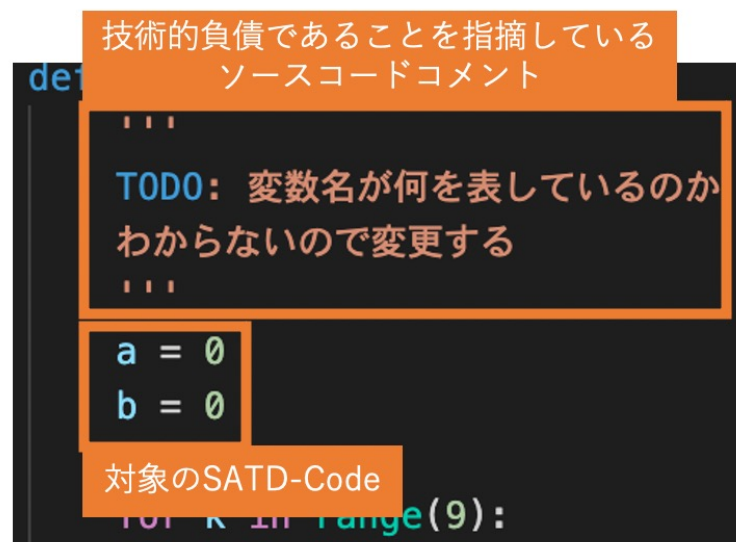
検出されたTD-Tool

- TD-Toolの検出のために, SonarQube¹を利用している

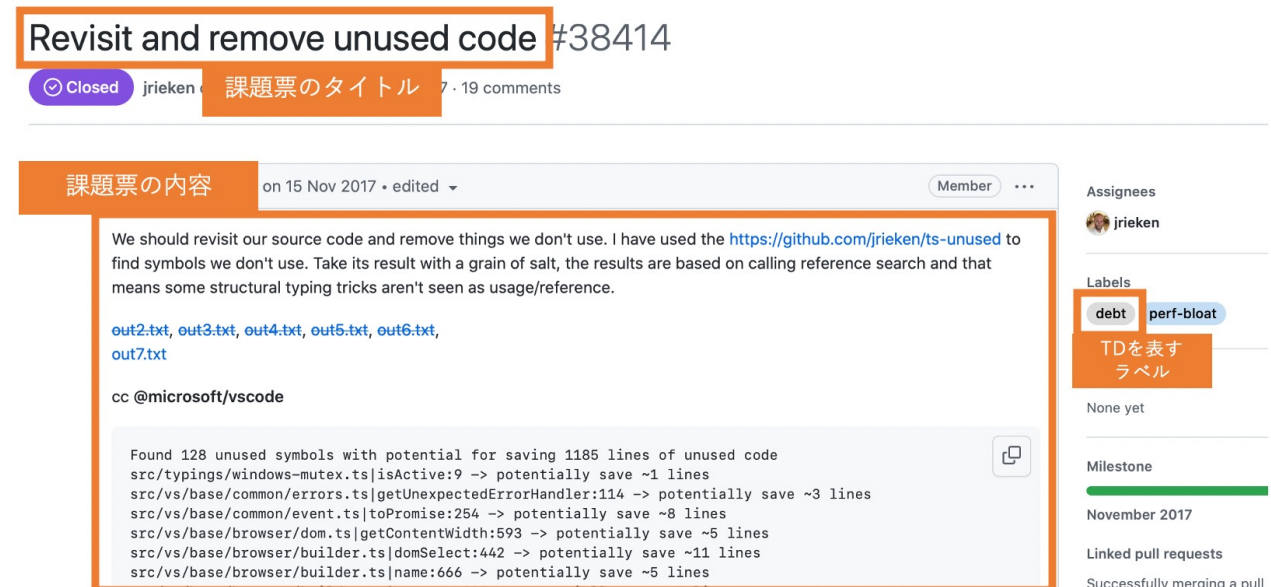
¹ <https://www.sonarqube.org/>

Self-Admitted TD (SATD)

- 開発者がTDであることを明文化しているTD
 - SATD-Code^[2]
 - ソースコードコメントを用いて明示しているSATD
 - SATD-Issue^[3]
 - 課題票にTDを表すラベルを付与することで明示しているSATD



SATD-Codeを指摘しているソースコードコメントと、指摘されたSATD-Code



SATD-Issueを指摘している課題票

[2] Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In 30th International Conference on Software Maintenance and Evolution (ICSME), pp. 91–100, 2014.

[3] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. Beyond the code: Mining self-admitted technical debt in issue tracker systems. MSR '20: Proceedings of the 17th International Conference on Mining Software Repositories, pp. 137–146, 2020.

背景

- プロダクトの保守性を高めるには、継続的に混入されるTDを返済する必要がある
- 開発には時間、人数などのリソースに限りがあるため、全てのTDを返済できない
- 限られたリソースで効率的にTDを返済するために、TDの優先順位付けに関する研究が盛んに行われている

背景

- 既存の優先順位付け手法
 - コスト・ベネフィット分析 (CBA)
 - ランキング
 - 予測分析
 - リアルオプション分析 (ROA)
 - 分析階層プロセス (AHP)
 - 現代ポートフォリオ理論 (MPT)
 - 加重和モデル (WSM)
 - ビジネスプロセス管理 (BPM)
 - 強化学習
 - ライフサイクル期待に基づくソフトウェア品質評価 (SQALE)

背景

- 既存の優先順位付け手法は、特定のTDに対応しているものが多く、SATD-IssueやSATD-Code, TD-ToolといったあらゆるTDに対応できない
- そのため、SATD-IssueやSATD-Code, TD-Toolに対応した優先順位付け手法を構築する必要がある

目的とアプローチ

- 目的
 - TD-Tool, SATD-Code, SATD-Issueの優先順位付け手法の構築
- アプローチ
 - 価値やコストを計算して比較を行う

研究の流れ (1/3)

- 既存の優先順位付け手法の調査（サーベイ）
 - どのような手法があって、どのようなTDに対応しているか、評価方法等を調べる
 - 価値, コスト, 資源制約とは？計算方法とは？
 - 優先順位付けできないTDとは？
 - どのように優先順位付け手法を評価している？
 - TDのどのような情報を用いて優先順位付けを行っているか？コード？テキスト？

研究の流れ (2/3)

- 優先順位付け手法の構築
 - 価値やコストの定式化？
 - なんで価値やコストを利用する必要がある？
 - 価値の高いTDから返済すべき？コストの低いTDから返済すべき？何を基準にする？
 - どうやって定式化する？何を利用する？
 - 計算に利用するメトリクスの選定？
 - TD-ToolやSATD-Code, SATD-Issueのどのような情報を用いて価値やコストを計算するのか？SATD-Codeならコメント？SATD-Issueならテキスト？コード？
 - 優先順位付け手法の選定？
 - 既存の手法である, CBAやランキング分析を使うのか？新たな手法で優先順位付けを行うのか？
 - 価値やコストの式をどのように活用するのか？

研究の流れ (3/3)

- 優先順位付け手法の評価実験
 - 実験の設定？
 - そもそもどうやって評価するのか？評価の妥当性は？
 - 対象プロジェクトの選定？
 - プロジェクト1つだけ？それとも複数？
 - TD-Toolの収集は時間がかかるけど、全部収集するのか？期間を決めて収集するのか？