情報処理学会論文誌 Vol.64 No.1 2–12 (Jan. 2023)

技術的負債に関連する課題票分類手法の構築

木村 祐太[1,a)] 大平 雅雄[1,b)]

受付日 2022年4月6日, 採録日 2022年10月4日

概要：技術的負債とは，短期的には有用だが長期的には問題を引き起こす実装や設計のことを指す．技術的負債には，開発者が明示的に管理を行う SATD という特殊な事象があり，課題票を用いて管理される事象を SATD-Issue という．SATD-Issue は OSS プロジェクトで一般的に扱われていないためデータの絶対数が少ない．そのため，SATD-Issue を対象とした研究を困難にしている．本論文では，SATD-Issue を対象とした研究を行う際に問題となるデータ不足の問題を解決する第 1 歩として，SATD-Issue の自動分類モデルの構築を行う．まず，SATD-Issue の特徴分析を行い，分類モデル構築に利用可能な特徴を明らかにする．次に，分析で得られた特徴を用いて構築したモデルの評価を行う．評価実験の結果，最も分類性能が良いモデルで Precision が 0.995，Recall が 0.903 となり，構築した分類モデルは SATD-Issue の分類が可能であることが分かった．このことから，SATD-Issue の自動分類モデルは技術的負債に関連する Issue の収集に役立てられると考える．

キーワード：技術的負債，課題管理システム，自動分類，機械学習

Building an Issue Classification Model on Technical Debt

Yuta Kimura[1,a)] Masao Ohira[1,b)]

Received: April 6, 2022, Accepted: October 4, 2022

Abstract: Technical debt (TD) refers to implementation and/or design that provides short-term benefit to a software project while it causes problems in the long run. Self-admitted technical debt (SATD) is a special case in which developers explicitly manages TD by themselves. SATD is classified into two types: SATD-comment and SATD-Issue. SATD-Comment is a clue of the existence of SATD that is expressed as a source code comment by individual developers. SATD-Issue is a SATD-related issue that is managed by a project using an issue tracking system. So far, SATD-Comment has been widely studied, but SATD-Issue has not been studied so much because few projects manage SATD at the project level. In this paper, we build a classification model of SATD-Issue as a first step to resolve the problem of the data shortage to be studied. First, we characterize SATD-Issue using a quantitative analysis to identify available features for building the classification model. Then, we evaluate the performance of the classification model built using the features identified by the analysis. As a result of our case study, our classification model shows 0.995 of Precision and 0.903 of Recall in the best case. From the result, we believe that our model would be useful to collect SATD-Issues from projects which do not explicitly manage technical debt.

Keywords: technical debt, issue tracking system, automated classification, machine learning

1. はじめに

近年，1 つのソフトウェアを長く運用し続けることが主流となりつつあり，ソフトウェアの大規模化が進んでいる．ソフトウェアは通常複数のソースコードファイルで構成されており，ソースコードファイル間には依存関係が存在する．ソフトウェアの大規模化により依存関係が高度に複雑になっているため，変更が与える他のソースコードファイルへの影響は大きくなっている．そのため，将来必要となる変更を容易に行えるよう，保守性を意識したソフトウェ

[1] 和歌山大学大学院システム工学研究科
Graduate School of Systems Engineering, Wakayama University, Wakayama 640–8510, Japan
[a)] kimura.yuta.1@g.wakayama-u.jp
[b)] masao@wakayama-u.ac.jp

ア開発が重要となっている．

技術的負債（Technical Debt）は，Cunningham によって初めて導入された，時間的制約とソフトウェア品質（特に保守性に関連する）のトレードオフについて言及したメタファ [7] として広く知られている．市場競争，リリース期限，ソフトウェアプロセスなどさまざまな要因から，開発者が最適ではない技術的な意思決定を下すことで技術的負債は導入される [16]．技術的負債の蓄積は保守コストの増加（新規機能の追加の妨げ，改修作業の複雑化）を招く [23], [26] ため，早期に技術的負債を返済する（技術的負債を取り除く）ことが望ましい．ただし，開発に利用できる資源（人的，金銭的，時間的）は有限であるため，技術的負債をどのように管理し影響を低減させるのかに注目する必要がある．そのため，実務者，研究者ともに技術的負債に対する関心が急速に高まっている [15]．

ほとんどの開発者が技術的負債を体系的・明示的に管理していない [3] 一方で，技術的負債のなかには，開発者が最適ではない技術的な意思決定を下したことを明文化する特殊な事象が存在する．このようなケースを Self-Admitted Technical Debt（SATD）[20], [24], [28] という．SATD には，技術的負債の存在をソースコード中にコメントとして開発者が個別に指摘する場合と，課題票（Issue）を用いて技術的負債が返済されるまでをプロジェクトとして管理する場合の 2 種類が存在する．本論文では，前者のソースコードコメントを用いた SATD を SATD-Comment，後者の Issue を用いた SATD を SATD-Issue と呼称する．特に本研究では，技術的負債に関連するラベル（“TD” など）がタグづけられた Issue を SATD-Issue と呼称することとする．

SATD-Comment は，オープンソースソフトウェア（OSS）プロジェクトにおいて広く見られる事象であることが明らかにされてきている [20]．SATD-Comment では主に，設計，要求，ドキュメント，テスト，欠陥に関する技術的負債が指摘される [24]．事前に設定されたルールに基づく既存の静的コード解析手法とは異なり，ソースコードコメントを用いることで開発者視点での技術的負債が検出できることを示している．静的コード解析による技術的負債の検出では，偽陽性が大量に出力されることがあり保守コストの抑制自体には効果的でないこともあるが，ソースコードコメントを用いた検出では，開発者が実際に返済の必要性を認識している技術的負債に焦点を絞った対応が可能となるため，近年では SATD-Comment に関する研究がさかんに行われている [24]．

一方，SATD-Issue は，プロジェクトとして意識的に技術的負債を管理するために，近年いくつかの OSS プロジェクトで運用され始めた段階にある [28]．SATD-Issue では，設計，UI，テスト，性能，インフラストラクチャ，ドキュメント，コード規約，ビルド，セキュリティ，要求など多岐にわたる技術的負債が指摘される [28]．課題票（Issue）を用いてプロジェクトが明示的に管理するため，Issue 解決までの過程で議論される技術的負債の混入原因，影響範囲，返済方法など，今後の技術的負債の管理支援に有益な情報が多く含まれているため，研究対象として注目されつつある．しかしながら，SATD-Comment に比べ SATD-Issue は OSS プロジェクトで一般的に扱われておらず，データの絶対数が少ないため SATD-Issue の研究を困難にしているという問題がある．実際に，GitHub にホスティングされているスター数上位 1,000 件の OSS プロジェクトを対象に調査したところ，19 件の OSS プロジェクト（1.9%）のみが SATD-Issue を扱っていることが分かった．さらに，10 件のプロジェクトのうち，open，closed 合わせて 100 件以上の SATD-Issue を管理している OSS プロジェクトは 3 件（0.3%）しか存在せず，現時点では研究対象として十分なデータが蓄積されていないことが分かった．一方で，技術的負債はある程度の年数が経過すれば多くのプロジェクトに存在すると考えるのが妥当であり，技術的負債に関連するラベルを付与せず明示されてはいないが Issue として技術的負債を扱っているプロジェクトは数多く存在するものと考えられる．

SATD-Issue を扱っていない OSS プロジェクトの課題管理システムのなかから，明示されてはいないが本質的には技術的負債と関連する Issue を抽出しデータを増やすことで，SATD-Issue を対象としたさまざまな研究の発展が見込める．たとえば，技術的負債に関連する Issue が抽出可能になることで，どのプロジェクトにどういった技術的負債が発生しやすいかの調査や，Issue で報告される技術的負債の影響の調査などが可能になる．また，Issue は解決時の変更を紐づけることができるため，プロジェクトが問題ととらえる技術的負債の実態を明らかにできる可能性がある．たとえば，これまでに研究されたきた技術的負債との比較調査により開発者・プロジェクトの観点と研究者の観点のギャップを明らかにすることができる．そのほかにも，SATD-Comment では扱えないプロジェクト視点での技術的負債の自動特定ツールの開発なども可能になると考える．

そこで本論文では，SATD-Issue を対象とした研究を行う際に問題となるデータ不足の問題を解決する第 1 歩として，SATD-Issue の自動分類モデルの構築を行う．自動分類モデルを構築できれば SATD-Issue を運用していないプロジェクトからもデータ収集が可能になると期待される．本論文では，主に 2 つの調査を行う．まず，(1) SATD-Issue を決定づける特徴の分析を行い，(2) (1) の分析で得られた特徴をもとに構築した分類モデルの評価を行う．(1) では，Issue が解決されるまでの工程に着目し関連する要素の特徴を抽出と仮説検定を用いた定量的分析を行う．(2) では，機械学習アルゴリズムを用いた分類モデルの構築と

分類性能の評価を行う．

本論文の貢献は以下の 3 つである．

- 報告者，プロセス，ソースコードの 3 つに関連する特徴から SATD-Issue を決定づける特徴を分析し，SATD-Issue に表れる特徴を明らかにした．
- SATD-Issue の自動分類モデルを構築し，最も分類性能が良いモデルでは Precision が 0.995，Recall が 0.903，F1 値が 0.947 となった．このことから，構築した分類モデルを利用することで SATD-Issue の分類が可能であることを示した．
- 分類性能の向上に寄与している特徴は報告者とプロセスに関する特徴であり，寄与していない特徴はソースコードに関する特徴であることを明らかにした．

以降の本論文の構成は次のとおりである．まず，調査対象のデータセットについて説明し，SATD-Issue の特徴分析を行う．その後，実験で利用する分類モデルの概要と評価実験について述べる．最後に，本論文の妥当性について議論し本論文をまとめる．

2. データセット

2.1 SATD-Issue の定義

本論文では，Issue を用いて文書化された技術的負債のことを SATD-Issue と呼ぶ．これは文献 [28] と同様の定義であり，特に，技術的負債に関連するラベルがタグづけられた Issue のことを指す．Issue では，時間経過とともに報告された問題の認識が変化することがある．たとえば，報告当初は技術的負債のラベルがつけられていたが，議論が重ねられることによってバグとして扱う方が適切であると判断され，Issue のラベルの付け替えが行われることがある．よって本論文では，問題内容が確定している（Issue のステータスが Closed），かつ，技術的負債に関連するラベルがタグづけられている Issue を SATD-Issue と見なす．

2.2 データの収集

本論文では，文献 [28] で言及されていた SATD-Issue を慣習的に取り扱っている OSS プロジェクトのうち，GitHub にホスティングされている 4 つの OSS プロジェクトのリポジトリ（**表 1**）を対象とする．対象とする Issue は，GitHub REST API を用いて 2021 年 3 月 30 日までにクローズされた Issue を対象のリポジトリから収集した．そして，SATD-Issue は技術的負債に関連するラベル（debt，technical debt）の有無によって判定している．最終的に収集されたデータについてまとめた結果を表 1 に示す．

表 1 対象リポジトリと Issue 数
Table 1 Studied repositories and # of issues.

リポジトリ	ラベル	**SATD**	その他
microsoft/vscode	debt	1,612	102,292
influxdata/influxdb	kind/tech debt	111	9,903
saleor/saleor	technical debt	106	2,623
nextcloud/server	technical debt	73	9,273

3. SATD-Issue の特徴分析

SATD-Issue の自動分類モデルの構築のために，SATD-Issue を決定づける特徴の分析を行う．まず，分析する報告者，テキスト，プロセス，ソースコードの 4 つのカテゴリについて説明し，その後，分析方法と結果について説明する．

3.1 報告者

報告者は開発に存在する潜在的な問題を発見し Issue として報告する開発者のことを指す．技術的負債の発見には，開発している製品への理解が関係すると想定される．開発されている製品を深く理解していない開発者が開発の障壁となる技術的負債の存在を認識できるとは考えられないため，製品を熟知している開発者が技術的負債を発見していると考えられる．また，SATD-Comment の研究においても，熟練の開発者であるほど SATD-Comment を導入する傾向にあることが明らかにされている [20]．これらのことから，開発プロジェクトに多く貢献している開発者やプロジェクト開発者であるかどうかが影響すると考えられるため，報告者の経験や種別を特徴量として利用する．

3.2 テキスト

テキストは報告される Issue に記述される自然言語のことを指す．SATD-Issue は，技術的負債ラベルがタグ付けられ他の Issue と区別し管理されている．つまり，開発者は他の Issue とは異なる事象として明確に認識し報告している．したがって，SATD-Issue と他の Issue では報告される内容に違いがあると考えられる．本論文では，Issue のタイトルと本文を特徴量として利用する．

3.3 プロセス

プロセスは Issue が報告されてから解決するまでに行われた活動に関わる情報のことを指す．技術的負債が保守コストを増加させることは広く知られている．また，他の Issue に比べて SATD-Issue は解決されるまでに時間がかかる [28]．くわえて，3.1 節で述べたように，技術的負債の認識には製品の理解が必要と考えられる．これらのことから，Issue に関与する開発者の数や変更ファイルの数，解決にかかった時間などを特徴量として利用する．また，SATD-Comment の研究 [18], [25] において，SATD-Comment を導入した開発者自身が返済する，つまり，自身の TODO タスクを管理するために SATD-Comment を導入する側面があることが明らかにされている．SATD-Issue でも同様の側面があると仮定し，報告者が自身を報告した Issue にアサ

表 2 特徴と分析結果
Table 2 Studied features and analysis result.

Category	Feature	p-value & Effect Size			
		vscode	influxdb	saleor	server
報告者 (7)	Experience（アカウント作成からの年月）	*M	*S	*S	*M
	OpenIssueNum（Issue の報告数）	*L	*M	*L	*L
	OpenPullRequestNum（プルリクエストの投稿数）	*L	*L	*L	*L
	rCommitNum（コミット数）	*L	*S	*S	*L
	Member（プロジェクトメンバかどうか）	*S	N	*S	*S
	Contributor（外部開発者かどうか）	*N	*N	N	N
	Collaborator（コラボレータかどうか）	N	N	N	―
テキスト (*)	Title（Issue のタイトル）	―	―	―	―
	Description（Issue の本文）	―	―	―	―
プロセス (9)	TitleLen（Issue のタイトルに含まれる単語数）	*N	N	*N	*S
	DescriptionLen（Issue の本文に含まれる単語数）	*M	*M	*N	*S
	AssigneeNum（Issue 担当者数）	N	*S	*S	*S
	SelfAssign（Issue 報告者が担当開発者かどうか）	*S	N	N	*N
	ParticipantNum（担当者を除く議論に参加した開発者数）	*M	*L	*S	N
	CommentNum（コメント数）	*S	*S	*S	*S
	pCommitNum（変更コミット数）	*L	N	N	N
	ChangeFileNum（変更ファイル数）	*L	N	N	N
	ResolutionTime（クローズされるまでにかかった時間）	*M	*M	*S	*L
ソースコード (14)	AddedClassNum（追加されたクラス数）	*S	―	―	*S
	DeletedClassNum（削除されたクラス数）	*S	―	―	N
	AddedFunctionNum（追加された関数・メソッド数）	*S	―	―	N
	DeletedFunctionNum（削除された関数・メソッド数）	*S	―	―	N
	AddedTotalLine（追加された変更行数）	*M	―	―	*M
	DeletedTotalLine（削除された変更行数）	*L	―	―	*S
	AddedLineOfCode（追加されたコード行数）	*S	―	―	S
	DeletedLineOfCode（削除されたコード行数）	*M	―	―	N
	AddedLineOfComment（追加されたコメント行数）	*S	―	―	N
	DeletedLineOfComment（削除されたコメント行数）	*M	―	―	N
	AddedEssential（増加した Essential 複雑度）	*S	―	―	N
	DeletedEssential（減少した Essential 複雑度）	*S	―	―	*S
	AddedMaxNesting（増加した最大ネスト数）	*S	―	―	S
	DeletedMaxNesting（減少した最大ネスト数）	*S	―	―	N

(*)：テキストの次元数はプロジェクトごとに異なる
*：$p < 0.01$，―：分析対象外
N：Negligible, S：Small, M：Medium, L：Large

インしているかどうかを表す *SelfAssign* を採用している．

3.4 ソースコード

ソースコードは Issue を解決するために行ったプロダクトへの変更のことを指す．従来より議論されてきた，技術的負債は製品に潜む問題（Code/Design TD）を指し示しており多く研究されている．たとえば，技術的負債の代表であるコードの臭いに関連する研究ではソースコードメトリクス[*1]を用いている [4], [10]．つまり，技術的負債の返済では何らかのソースコードの変化が生まれると想定する．本論文では，ソースコードの変化をソースコードメトリクスとして計測する．ソースコードメトリクスの計測では，Issue ID を含むコミットを対象に行う．そのコミット前後でのリビジョンのソースコードメトリクスを計測し，その差分を特徴としてとらえる．特徴には，コードの追加/削除行数や複雑度の変化を用いる．

以上，4 つのカテゴリに含まれる具体的な特徴量の一覧を**表 2** に示す．

3.5 方法

3.5.1 分析対象とする特徴

Issue の分類に関連する従来研究 [22], [27] ではテキスト

[*1] ソースコードに関連する特徴全般のことを指し，コードの変更行数や複雑度などが含まれる．

をベースとした分類モデルが提案されている．本論文でも同様に，テキストをベースとした分類モデルの構築を行う．今回は，テキスト以外に分類性能に寄与する特徴を理解するために，報告者，プロセス，ソースコード，それぞれ 3 つの要素の特徴を対象に分析を行う．

3.5.2 仮説検定と効果量測定

SATD-Issue の特徴を理解するための方法として仮説検定を行う．仮説検定の結果，もし有意差（$p < 0.01$）が見られれば，SATD-Issue を特徴づけるためにその特徴が利用できると考える．また，その特徴において SATD-Issue とそのほかの Issue がどの程度異なるのかを効果量を用いて定量的に示す．前節で述べた特徴には 2 種類（質的，量的）存在するため，それぞれ異なる検定手法と効果量を用いる．*Type* や *SelfAsign* など 2 値で表される質的特徴ではカイ二乗検定と *Choen's w* を利用し，*OpenIssueNum* や *CommentNum* など連続値で表される量的特徴ではマンホイットニーの U 検定と *Cliff's delta* を利用する．

3.6 結果

分析結果を表 2 に示す．有意差が見られた特徴にはアスタリスクを付与し，さらに，効果量が Small 以上の場合には効果量の大きさを S（Small），M（Medium），L（Large）で区別できるようにしている．3.5.1 項で述べたように，今回の分析ではテキスト以外に分類性能の向上に影響しそうな特徴を明らかにすることを目的とするため，テキストの分析を除外している．また，サンプル数が 10 以下の特徴に関しては，仮説検定に耐えうると想定できないため除外している．分析から除外している特徴には斜線を引いている．

すべてのプロジェクトで共通して，報告者の開発経験を表す *Experience*，*OpenIssueNum*，*OpenPullRequestNum*，*rCommitNum*，Issue に言及したコメント数を表す *CommentNum*，クローズされるまでにかかった時間を表す *ResolutionTime* で有意差が確認された．なかでも，*OpenIssueNum* や *OpenPullRequestNum* といった開発への貢献に関する特徴でより強い効果量が見られた．

4. 分類モデル

本研究ではプロジェクトごとに分類モデルを構築する．収集したデータを統合してより汎用性の高いモデルを構築する方法も考えられるが，(1) プロジェクトごとに開発する製品が異なるため扱う技術的負債が異なる可能性が高いこと，(2) 表 2 の結果より同じ特徴量であっても効果量が大きく異なることがある一方で，表 1 より vscode のデータが他を圧倒しており vscode の特徴が分類モデルに強く反映される懸念が高いことから，データセットを統合した分類モデルは構築せずプロジェクトごとに分類モデルを構築することとした．

4.1 テキストの前処理

特定単語の置換：テキストには学習のノイズとなる文字列が含まれるため，特定の単語への置換を行う．今回は，コードスニペット，インラインコード，画像，URL，ファイル名などを正規表現を用いて特定の単語（codesnippet，filename など）に置換する．

単語分割：テキストを単語単位に分割する．分割するための Tokenizer には spaCy [*2] を利用する．Tokenizer に使用するコーパスは `en_core_web_lg` を用いる．spaCy の Tokenizer [*3] は，まず，空白文字でテキストを分割する．そして，分割した各テキストにクエスチョンマークやアポストロフィなどが含まれる場合はさらに分割を行う．

ストップワードと Markdown タグの除去：テキストを用いた分類では，ストップワードがノイズとなることが知られている．そのため，本論文でも “I”，“You”，“a” などのストップワードを除外する．また，Markdown タグは Issue 内容を表す情報ではなくノイズとなるため除外する．

固有名詞と未登録語の置換：spaCy の POS-tagger を用いて各単語の品詞の検出を行い，固有名詞と Out-of-Vocabulary（OOV）と判定された単語を uniqueword や oov という単語にそれぞれ置換する．OOV は，spaCy のコーパスに記録されていない単語のことを示す．固有名詞と OOV の置換は，固有名詞など各プロジェクト特有の表記を学習することで起こる過学習を抑えることが目的である．

単語の正規化：分割された単語は，大文字や小文字，3 人称単数や複数形の s など表記揺れが存在する．このとき，同じ意味を表す単語であっても同一のものと見なすことができないため，単語の正規化を行う必要がある．単語の正規化では，まず，すべての単語を小文字に統一し，数字を 0 に統一する．そして，Lemmatization（見出語への変換）を行い表記を統一する．

ベクトル化：テキストのベクトル化では，Bag-of-Words（BoW）と TF-IDF を利用する．BoW は，コーパスを用いて辞書を作成し各テキストでの単語の出現頻度をベクトルとして扱う手法である．TF-IDF は，ある単語の出現頻度 TF とある単語を含む文書の逆出現頻度 IDF を用いて BoW で生成されたベクトルに重み付けをする手法である．

4.2 機械学習アルゴリズム

モデルの学習には，分類タスクで一般的に用いられている機械学習アルゴリズムであるランダムフォレスト [13]，ロジスティック回帰 [2]，Support Vector Machine（SVM）[6] を用いる．学習時のハイパーパラメータに関して，ランダムフォレストは木の本数を 200，ノードの分岐に使用する特徴の個数を全特徴の個数の平方根，ロジスティック回帰は正則化項を L2 ノルム，反復回数を 2000，SVM は正則

[*2] https://spacy.io
[*3] https://spacy.io/usage/spacy-101#annotations-token

化項を L2 ノルム，反復回数を 2000 とし，ランダムシードはすべてのアルゴリズムで 42 に設定した．そのほかのパラメータに関してはデフォルト値を採用している．

4.3 正規化と欠損値補完

モデルの学習にはさまざまな特徴を使用するが，特徴ごとに尺度（単位）が異なる．ランダムフォレストは尺度の違いによる影響が少ないが，線形モデルであるロジスティック回帰や SVM などは尺度の違いによって正しく学習できないことがある．そこで，本実験では 0 から 1 の範囲に値を正規化（Min-Max 法）する．また，モデルに組み込む特徴には欠損値を含む特徴が存在するため欠損値への対処が必要とされる．欠損値を含むデータを除外するリストワイズ法を用いた場合，貴重なデータが損失しモデルが十分に学習できない恐れがある．そのため，本実験では，回帰代入法を用いて欠損値の補完を行う．具体的には，Lasso 回帰を用いて欠損値を含まない特徴を説明変数として学習し欠損値を含む特徴を目的変数として値の予測と補完を行う．

4.4 スクリプトとデータセットの公開

本論文で使用する分類モデル構築のスクリプトとデータセットは GitHub 上に公開している[*4]．

5. 評価実験

前章で構築した分類モデルの評価実験を行う．構築する分類モデルの分類性能を明らかにすることで，SATD-Issue が報告される際にモデル構築に利用した特徴量（表 2）がどの程度寄与しているのか（あるいは，考慮できていない特徴量が存在するのか）かをプロジェクトごとに評価する．分類性能が高いほどプロジェクトにおける報告される SATD-Issue の特徴が定まっていると解釈できる．

実験方法として，分析で得られた特徴から設定する基準を満たす特徴を用いてプロジェクトごとに分類モデルを構築し分類性能の評価を行う．評価では層化 5 分割交差検証を実施する．ただし，最初のデータ分割による精度への影響があるため，影響低減のために層化 5 分割交差検証を 20 回行う計 100 回の平均分類性能を評価する．

5.1 モデル入力に用いる特徴

SATD-Issue の特徴分析で有意差が見られ効果量が Small 以上の特徴をモデルの構築に利用する．ただし，モデルの学習に線形アルゴリズムを利用するため多重共線性を考慮する必要がある．多重共線性とは，説明変数間の相関の強さのことをいう．ロジスティック回帰などでは説明変数が独立していることを仮定しており，多重共線性が高い説明変数が含まれる場合ではモデルの予測性能が劣化することが知られている．本論文では，多重共線性の指標として Variance Inflation Factor（VIF）を利用する．VIF の計算は次のとおりである．まず，すべての特徴のなかから 1 つの特徴を目的変数としそのほかすべての特徴を説明変数として最小二乗回帰を繰り返し行う．ある特徴 X_i を目的変数としたときの最小二乗回帰は以下の式で表される．このとき c_0 は定数，e は誤差を表す．

$$X_i = \alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_k X_k + c_0 + e$$
$$(k = 1, 2, \cdots, i, \cdots, n)$$

最小二乗回帰から得られる決定係数 R_i^2 を用いることで，VIF は以下の式で表される

$$VIF_i = \frac{1}{1 - R_i^2}$$

算出された VIF が最も大きい値となる特徴を削除し，すべての特徴において得られた VIF が 3 未満となるまで繰り返し計算と特徴の削除を行う．最終的にモデルの構築に利用する特徴は表 2 において灰色で強調している．

表 3 データセット（外れ値除去後）
Table 3 Dataset without outliers.

リポジトリ	SATD	その他
microsoft/vscode	958	50,625
influxdata/influxdb	71	6,369
saleor/saleor	86	2,019
nextcloud/server	55	4,819

5.2 データセットの再構築

モデルの構築時に外れ値を含むデータを学習するとモデルが歪められてしまう（正常なデータを正しく予測分類できなくなる）ため，あらかじめデータセットから外れ値を含むデータを除外する必要がある．そこで，5.1 節で選択した特徴をもとに外れ値を含むデータを除外する．今回使用する各特徴は正規分布が仮定できないため，外れ値の判定には箱ひげ図の四分位範囲を用いる．

外れ値を含むデータの除外は前述の外れ値判定に基づいており，外れ値を 1 つでも含むデータをデータセットから除外する．外れ値を含むデータを除外した最終的なデータセットを**表 3** に示す．

5.3 評価指標

分類性能の評価指標には，Precision, Recall, F1 値, Accuracy を用いる．SATD-Issue を SATD-Issue と予測したものを真陽性（TP：True Positive），その他の Issue を SATD-Issue と予測したものを偽陽性（FP：False Positive），その他の Issue を Issue をその他の Issue と予測したものを真陰性（TN）SATD-Issue をその他の Issue と予測したものを偽陰性（FN：False Negative）とすると，Precision と

[*4] https://github.com/kyuta8/SATDIssueClassification

表 4 モデルの分類性能
Table 4 Classification performance of our model.

Project		ロジスティック回帰				ランダムフォレスト				SVM			
		Pre.	**Rec.**	**F1**	**Acc.**	**Pre.**	**Rec.**	**F1**	**Acc.**	**Pre.**	**Rec.**	**F1**	**Acc.**
vscode	BoW	0.983	0.837	0.904	0.997	1.000	0.862	0.925	0.998	0.982	0.895	0.936	0.998
	TF-IDF	0.988	0.831	0.903	0.997	1.000	0.889	0.941	0.998	**0.995**	**0.903**	**0.947**	0.998
influxdb	BoW	0.795	0.092	0.162	0.990	0.820	0.085	0.151	0.990	**0.762**	**0.238**	**0.352**	0.991
	TF-IDF	0.160	0.011	0.021	0.989	0.800	0.082	0.146	0.990	0.820	0.148	0.242	0.990
saleor	BoW	0.589	0.060	0.107	0.961	0.360	0.022	0.041	0.960	**0.475**	**0.213**	**0.285**	0.958
	TF-IDF	0.020	0.001	0.002	0.959	0.190	0.011	0.021	0.960	0.674	0.103	0.174	0.962
server	BoW	1.000	0.398	0.553	0.993	0.880	0.176	0.284	0.991	0.962	0.521	0.667	0.994
	TF-IDF	1.000	0.415	0.575	0.993	0.930	0.199	0.318	0.991	**0.965**	**0.525**	**0.671**	0.994

Recall は以下の式で表される．

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Precision と Recall は 0 から 1 の値をとり，予測した Issue すべてが SATD-Issue だった場合に Precision は 1 となり，すべての SATD-Issue が予測できた場合に Recall は 1 となる．また，F1 値は Precision と Recall の調和平均を用いて以下の式で表される．

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

Accuracy は TP，FP，TN，FN を用いて以下の式で表される．構築したモデルが誤分類しなければ Accuracy は 1 となる．

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

5.4 実験結果

各分類モデルの分類性能の評価結果を**表 4** に示す．各プロジェクトにおいて最も高くなった Precision，Recall，F1 値を太字で強調する．評価結果の解釈では，ある程度の誤検出を許容しより多くの SATD-Issue を検出することを重視する観点（Recall 優先）で行う．そのため，F1 値が太字で強調されている分類モデルに着目する．

microsoft/vscode：TF-IDF と SVM を用いたモデルが最も分類性能が良いモデルとなり，Precision が 0.995，Recall が 0.903，F1 値が 0.947 であった．このことから，vscode では SATD-Issue とその他の Issue の特徴にはっきりと違いがあることが分かる．vscode は最も SATD-Issue を保有しているプロジェクトであることからも，プロジェクトとして Issue として扱う技術的負債を徹底しているのではないかと考えられる．つまり，vscode のデータを学習した分類モデルをその他のプロジェクトに適用することで，vscode に報告される SATD-Issue の特徴に沿った技術的負債に関連する Issue が抽出できる．

influxdata/influxdb：BoW と SVM を用いたモデルが最も分類性能が良いモデルとなり，Precision で 0.762，Recall が 0.238，F1 値が 0.352 であった．Recall が 0.238 と低いため，SATD-Issue とその他の Issue の特徴にはっきりとした違いがあるわけではない．ただし，Precision が 0.762 と高い値を示していることから，一部の SATD-Issue では特徴に違いがあることが分かる．研究者が分析対象データとして SATD-Issue を運用していないプロジェクトから技術的負債を扱っている Issue を収集するために本モデルを適用することを想定した場合，実用的な観点では分類モデルの性能は高くないため，さらなる分類性能の向上が必要であると考える．

saleor/saleor：BoW と SVM を用いたモデルが最も分類性能が良いモデルとなり，Precision が 0.475，Recall が 0.213，F1 値が 0.285 であった．Precision，Recall ともに低いことから，SATD-Issue とその他の Issue の特徴に違いがないことが分かる．したがって，saleor のデータを学習した分類モデルの実用性は低い．

nextcloud/server：TF-IDF と SVM を用いたモデルが最も分類性能が良いモデルとなり，Precision が 0.965，Recall が 0.525，F1 値が 0.671 であった．おおよそ半数の SATD-Issue を正しく分類できていることから，SATD-Issue とその他の Issue の特徴に違いがあることが分かる．Recall を向上させることができれば，分類モデルの実用性をさらに高めることができる．

> vscode のデータを学習した分類モデルの分類性能が最も良く，F1 値が 0.947 であることから実用性も高いことが分かった．influxdb，server のデータを学習した分類モデルは Precision が高い（0.762，0.965）が Recall が低い（0.238，0.352）ため，さらなる分類性能の向上のための工夫が必要である．saleor のデータを学習した分類モデルは Precision，Recall ともに低く実用性が乏しいことが分かった．

5.5 モデルの誤分類

SATD-Issue を運用していないプロジェクトから技術的

表 5 アブレーションスタディの結果（vscode）
Table 5 Result of ablation study for vscode.

Model	Pre.		Rec.		F1	
T+R+P+C	*0.995	(+14.3%)	*0.903	(+44.1%)	*0.947	(+34.9%)
T+R+P	*0.996	(+14.4%)	*0.903	(+44.1%)	*0.947	(+34.9%)
T+R+C	*0.995	(+14.2%)	*0.795	(+33.4%)	*0.884	(+28.5%)
T+P+C	*0.958	(+10.6%)	*0.800	(+33.8%)	*0.872	(+27.3%)
T+R	*0.994	(+14.2%)	*0.801	(+34.0%)	*0.887	(+28.9%)
T+P	*0.957	(+10.5%)	*0.799	(+33.7%)	*0.871	(+27.2%)
T+C	0.850	(−0.2%)	0.458	(−0.4%)	0.594	(−0.4%)
T	0.852		0.462		0.598	

T：テキスト，R：報告者，P：プロセス，C：ソースコード
*：$p < 0.01$

負債に関連する Issue を抽出しデータを増やすことを想定した場合，SATD-Issue を運用していないプロジェクトにおいて分類モデルは実際にどのくらいの性能を発揮できるのかを理解することは重要である．しかし，SATD-Issue を運用していないプロジェクトを用いた評価において妥当な評価方法が存在しない．そこで，本論文では分類モデルの Accuracy に着目する．分類モデルによる Issue の誤分類がどのくらい存在するのかを示すことで，データを増やす際のノイズとなる技術的負債に関連しない Issue がどのくらい存在しそうかを理解することができると考える．

表 4 の Accuracy に注目すると，すべての分類モデルにおいて高い値（0.998 から 0.958）であることが分かる．5.4 節では，influxdb や saleor の分類モデルの分類性能が低いことが分かった．しかし分類性能が低かった influxdb の分類モデルでは Accuracy が 0.991，saleor の分類モデルでは Accuracy が 0.958 であることが分かる．このことから，本論文が構築した分類モデルは誤分類が少なく，データを増やしてもノイズをあまり含まないと考えられる．

6. 性能向上に寄与している特徴は何か？

構築したモデルにおいて，分類性能の向上に寄与している特徴を具体的に明らかにするためにアブレーションスタディを行う．アブレーションスタディとは，モデルに含まれる要素を削除することで，その要素がモデルの性能にどのくらい貢献しているのかを評価する方法のことをいう．アブレーションスタディでは，評価実験で最も良い分類性能を示した vscode の分類モデル（TF-IDF と SVM を用いたモデル）を利用し，報告者，プロセス，ソースコードの特徴を削除したモデルの分類性能を示す．また，すべてのカテゴリの特徴を削除し，テキストの特徴のみを利用した分類モデルをベースラインとする．評価方法は，前節の評価実験と同様，層化 5 分割交差検証を 20 回行い，そのときの Precision，Recall，F1 値の平均で評価する．また，ベースラインとの比較として，ベースラインの分類性能からの向上率の算出とマンホイットニーの U 検定を用いた分類性能の有意差検定（$p < 0.01$）を行う．

アブレーションスタディの結果を**表 5** に示す．表中のモデルは分類モデルに利用しているカテゴリの特徴の組合せを表しており，各評価指標の項目に含まれる括弧中の値はベースラインと比較しどのくらい向上したかを表す．分類性能の向上を直感的に理解するために，括弧中の値は百分率で示している．また，ベースラインと比較し分類性能が優位に異なったものにはアスタリスクをつけている．まず，ベースラインであるテキストの特徴のみを利用した分類モデル（T）の分類性能に注目すると，Precision が 0.852，Recall が 0.462 と比較的高い性能で分類できていることが分かる．次に，ベースラインに報告者，プロセス，ソースコードのいずれか 1 つのカテゴリの特徴を加えた分類モデルに注目すると，報告者の特徴を加えた分類モデル（T + R）とプロセスの特徴を加えた分類モデル（T + P）においてベースラインの分類性能との間に有意差が見られている．一方で，ソースコードの特徴を加えた分類モデル（T + C）はベースラインの分類性能との間に有意差が見られなかった．このことから，ソースコードの特徴が分類性能の向上に寄与しないことが分かった．次に，ベースラインに報告者，プロセス，ソースコードのいずれか 2 つのカテゴリの特徴を加えた分類モデルに注目すると，プロセスとソースコードの特徴を加えた分類モデル（T + P + C）が最も性能の向上率が低いことが分かる．このことから，報告者の特徴が分類性能の向上に最も寄与している特徴であることが分かった．

7. 妥当性への脅威

7.1 内的妥当性

SATD-Issue の定義：本論文では，SATD-Issue 特定のために Issue のラベル機能を用いている．特定には “debt” や “technical debt” のラベルを使用しているが，そのほかに技術的負債を表すラベルが存在する可能性がある．

テキストのベクトル化：本論文では，分類モデルの構築に用いるテキストの特徴を BoW と TF-IDF を用いてベクト

ル化している．そのため，テキストに出現する単語の有無などの情報しか用いていない．最近の自然言語処理技術では，テキストの意味的情報（文脈など）を組み込むために，ニューラルネットワークを用いた情報埋め込みを利用している技術が多く存在する（Word2Vec や Transformer など）．これらの技術を用いてテキストのベクトル化を行うことで分類性能が向上する可能性がある．しかし，SATD-Issue の自動分類はこれまでに行われていないため，本論文の位置づけとして，まずは比較対象となるベースラインの構築をすることにある．

ソースコードの情報取得と利用した特徴：本論文で用いているコードの情報は，Issue の ID を含むコミット（変更）を用いて取得している．しかし，Issue の解決にはプルリクエストを用いて行われることがあるため，プルリクエストを介した結果を含めた場合には結果が異なる可能性がある．しかし，直接 Issue の ID が紐づけられていないコミットを対象とすると Issue の解決以外の内容を含む可能性があるため，必ずしも良い結果につながるとは限らない．

7.2 外的妥当性

本論文では，4 つの OSS プロジェクトを対象に調査を行っている．対象としているプロジェクトが少ないため，本論文で得られた結果に一般性があるかは定かでない．しかし，対象となった 4 つの OSS プロジェクトは異なるドメインのプロジェクトであるため，外的妥当性への脅威はある程度緩和されていると考える．

8. 関連研究

8.1 技術的負債検出に関連する研究

従来の調査により，技術的負債の蓄積が開発に否定的な影響を与えることが明らかにされている．そのため，開発を支援するための技術的負債の検出に関する研究がさかんに行われている．検出方法としては，主に 2 つの方法が存在する．1 つは，コード解析を用いた技術的負債の検出方法である [4], [14], [29]．もう 1 つは，依存関係解析を用いた技術的負債の検出方法である [5], [19]．これらの方法では，ソースコードメトリクスを用いて計測した値による判定やパターンベースの違反検知などを用いている．最近では，機械学習による検出方法 [1], [11] も多く提案されている．

8.2 SATD-Comment 検出に関連する研究

Potdar ら [20] は，4 つの OSS システムから 101,762 のソースコードコメントを抽出し手動で分析している．分析の結果，SATD-Comment を示す 62 のコメントパターン（*hack*，*fixme* など）を明らかにしパターンベースでの検出を可能にした．文献 [20] のパターンベースの検出アプローチを拡張する目的として，Farias ら [8] は，Contextualized Vocabulary Model（CVM）の提案を行っている．そのほかにも，手動でのコメント調査を自動化するために，テキストマイニングを用いた研究 [12], [17] も存在する．さらに最近では，SATD-Comment 検出手法の向上のために Natural Language Processing（NLP）や機械学習を用いたアプローチが提案されている [9], [18], [21].

9. おわりに

技術的負債の蓄積によりソフトウェア開発の保守コストは増加する．それゆえ，技術的負債を管理し適切に返済する必要がある．しかし，技術的負債の返済には多くの時間を必要とするため，開発者が技術的負債を効率的に返済するための支援を必要とする．開発者への支援のためには，まず開発者が問題として認識している技術的負債の実態を明らかにしなければならない．SATD-Issue は，技術的負債の返済に至るまでの議論や実際に行ったコードの変更が取得できる．SATD-Issue を用いることで，技術的負債の返済のための知見が得られると考える．しかし，SATD-Issue は OSS プロジェクトにおいて一般的に扱われていないためデータの絶対数が少なく研究を困難にしている．この問題に対して，SATD-Issue を扱っていない OSS プロジェクトの課題管理システムのなかから，明示されてはいないが本質的には技術的負債と関連する Issue を抽出しデータを増やすことで問題解決を図る．本論文では，SATD-Issue の問題解決の第 1 歩として，SATD-Issue の自動分類モデルの構築を行った．主に 2 つの調査を行い，(1) SATD-Issue を決定づける特徴の分析と，(2) (1) の分析で得られた特徴をもとに構築した分類モデルの評価実験を行った．評価実験では，分類性能が良いモデルがいくつか見られ，vscode の分類モデルが最も分類性能が良く Precision が 0.995，Recall が 0.903，F1 値が 0.947 であった．また，アブレーションスタディを行い分類性能の向上に寄与している特徴の調査を行ったところ，報告者の特徴が最も寄与しており，ソースコードの特徴が最も寄与していないことが分かった．

今後の予定としては，技術的負債を明示的に管理していないプロジェクトへ構築した分類モデルを適用し，分類モデルの一般性および有用性を評価すること，分類モデルのベースラインとなっているテキスト特徴量を Word2Vec や Transformer など最新のベクトル化技法を用いてさらなる分類性能の向上を目指すことなどがあげられる．

謝辞 本研究の一部は，文部科学省科学研究補助金（基盤（C）：22K11974）による助成を受けた．

参考文献

[1] Amorim, L., Costa, E., Antunes, N., Fonseca, B. and Ribeiro, M.: Experience Report: Evaluating the Effectiveness of Decision Trees for Detecting Code Smells, *Proc. 26th International Symposium on Software Reliability Engineering* (*ISSRE'15*), pp.261–269 (2015).

[2] Berkson, J.: Application of the Logistic Function to Bio-Assay, *Journal of the American Statistical Association*, Vol.39, No.227, pp.357–365 (1944).
[3] Bogner, J., Fritzsch, J., Wagner, S. and Zimmermann, A.: Limiting technical debt with maintainability assurance: an industry survey on used techniques and differences with service- and microservice-based systems, *Proc. 2018 International Conference on Technical Debt* (*TechDebt'18*), pp.125–133 (2018).
[4] Bohnet, J. and Döllner, J.: Monitoring Code Quality and Development Activity by Software Maps, *Proc. 2nd Workshop on Managing Technical Debt* (*MTD'11*), pp.9–16 (2011).
[5] Brondum, J. and Zhu, L.: Visualising Architectural Dependencies, *Proc. 3rd International Workshop on Managing Technical Debt* (*MTD'12*), pp.7–14 (2012).
[6] Cortes, C. and Vapnik, V.: Support-Vector Networks, *Machine Learning*, Vol.20, pp.273–297 (1995).
[7] Cunningham, W.: The WyCash Portfolio Management System, *SIGPLAN OOPS Mess.*, Vol.4, No.2, pp.29–30 (1992).
[8] de Freitas Farias, M.A., Santos, J., Kalinowski, M., de Mendonça Neto, M.G. and Spínola, R.O.: Investigating the Identification of Technical Debt Through Code Comment Analysis, *Proc. 18th International Conference on Enterprise Information Systems* (*ICEIS'16*), pp.284–309 (2016).
[9] Flisar, J. and Podgorelec, V.: Identification of Self-Admitted Technical Debt Using Enhanced Feature Selection Based on Word Embedding, *IEEE Access*, Vol.7, pp.106475–106494 (2019).
[10] Fontana, F.A., Ferme, V. and Spinelli, S.: Investigating the Impact of Code Smells Debt on Quality Code Evaluation, *Proc. 3rd International Workshop on Managing Technical Debt* (*MTD'12*), pp.15–22 (2012).
[11] Fontana, F.A., Zanoni, M., Marino, A. and Mäntylä, M.V.: Code Smell Detection: Towards a Machine Learning-Based Approach, *Proc. 29th International Conference on Software Maintenance* (*ICSM'13*), pp.396–399 (2013).
[12] Haung, Q., Shihab, E., Xia, X., Lo, D. and Li, S.: Identifying Self-Admitted Technical Debt in Open Source Projects Using Text Mining, *Empirical Software Engineering*, Vol.23, No.1, pp.418–451 (2018).
[13] Ho, T.K.: Random Decision Forests, *Proc. 3rd International Conference on Document Analysis and Recognition* (*ICDAR'95*), pp.278–282 (1995).
[14] Letouzey, J.-L. and Ilkiewicz, M.: Managing Technical Debt with the SQALE Method, *IEEE Software*, Vol.29, No.6, pp.44–51 (2012).
[15] Li, Z., Avgeriou, P. and Liang, P.: A Systematic Mapping Study on Technical Debt and its Management, *Journal of Systems and Software*, Vol.101, pp.193–220 (2015).
[16] Lim, E., Taksande, N. and Seaman, C.: A Balancing Act: What Software Practitioners Have to Say about Technical Debt, *IEEE Software*, Vol.29, No.6, pp.22–27 (2012).
[17] Liu, Z., Huang, Q., Xia, X., Shihab, E., Lo, D. and Li, S.: SATD Detector: A Text-Mining-Based Self-Admitted Technical Debt Detection Tool, *Proc. 40th International Conference on Software Engineering: Companion* (*ICSE-Companion'18*), pp.9–12 (2018).
[18] Maldonado, E.d.S., Shihab, E. and Tsantalis, N.: Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt, *IEEE Trans. Softw. Eng.*, Vol.43, No.11, pp.1044–1062 (2017).
[19] Mo, R., Garcia, J., Cai, Y. and Medvidovic, N.: Mapping Architectural Decay Instances to Dependency Models, *Proc. 4th International Workshop on Managing Technical Debt* (*MTD'13*), pp.39–46 (2013).
[20] Potdar, A. and Shihab, E.: An Exploratory Study on Self-Admitted Technical Debt, *Proc. 30th International Conference on Software Maintenance and Evolution* (*ICSME'14*), pp.91–100 (2014).
[21] Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X. and Grundy, J.: Neural Network-Based Detection of Self-Admitted Technical Debt: From Performance to Explainability, *ACM Trans. Software Engineering and Methodology*, Vol.28, No.3, pp.1–45 (2019).
[22] Runeson, P., Alexandersson, M. and Nyholm, O.: Detection of Duplicate Defect Reports Using Natural Language Processing, *Proc. 29th International Conference on Software Engineering* (*ICSE'07*), pp.499–510 (2007).
[23] Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y. and Vetrò, A.: Using Technical Debt Data in Decision Making: Potential Decision Approaches, *Proc. 3rd International Workshop on Managing Technical Debt* (*MTD'12*), pp.45–48 (2012).
[24] Sierra, G., Shihab, E. and Kamei, Y.: A Survey of Self-Admitted Technical Debt, *Journal of Systems and Software*, Vol.152, pp.70–82 (2019).
[25] Tan, J., Feitosa, D. and Avgeriou, P.: An Empirical Study on Self-Fixed Technical Debt, *Proc. 3rd International Conference on Technical Debt* (*TechDebt'20*), pp.11–20 (2020).
[26] Tom, E., Aurum, A. and Vidgen, R.: An Exploration of Technical Debt, Vol.86, No.6, pp.1498–1516 (2013).
[27] Wang, X., Zhang, L., Xie, T., Anvik, J. and Sun, J.: An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information, *Proc. 30th International Conference on Software Engineering* (*ICSE'08*), pp.461–470 (2008).
[28] Xavier, L., Ferreira, F., Brito, R. and Valente, M.T.: Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems, *Proc. 17th International Conference on Mining Software Repositories* (*MSR'20*), pp.137–146 (2020).
[29] Zazworka, N., Vetrò, A., Izurieta, C., Wong, S., Cai, Y., Seaman, C. and Shull, F.: Comparing four Approaches for Technical Debt Identification, *Software Quality Journal*, Vol.22, pp.1–24 (2013).

木村 祐太

令和 2 年和歌山大学システム工学部システム工学科卒業．令和 4 年和歌山大学大学院システム工学研究科博士前期課程修了．令和 4 年より和歌山大学大学大学院システム工学研究科博士後期課程（在学中）ソフトウェア工学，特にマイニングソフトウェアリポジトリの研究に従事．

大平 雅雄 （正会員）

平成 10 年京都工芸繊維大学工芸学部電子情報工学科卒業，平成 15 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同大学情報科学研究科助教を経て，平成 24 年和歌山大学システム工学部准教授．博士（工学）．ソフトウェア工学，特にマイニングソフトウェアリポジトリの研究に従事．電子情報通信学会，IEEE，ACM 各会員．