Vol.2022-SE-210 No.10
2022/3/11

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)]

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おこなう 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．本論文では，SATD-Issue 自動分類モデルの構築をおこなう．SATD-Issue の特徴分析をおこない，分類モデル構築に利用可能な特徴を明らかにする．また，分析で得た特徴を用いて構築したモデルの評価をおこなう．評価実験の結果，最も分類精度が良いモデルで Precision が 0.995，Recall が 0.903 となった．また，アブレーションスタディの結果，報告者に関する特徴が最も分類に寄与していることがわかった．

Yuta Kimura[1,a)]　Masao Ohira[1,b)]

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，それにともなったソフトウェアの大規模化が進んでいる．ソフトウェアは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ソースコードファイルの依存関係がさらに複雑になり，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，将来的な保守を容易におこなうためのソフトウェア開発（保守性を意識した開発）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 [6] として広く知られている．市場競争，リリース締切，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 [15]．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 [22, 24] 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．

前述の状況から，ソフトウェアの大規模化によって技術的負債が与えるの開発（保守コスト）への影響が大きくなっている．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を集めることとなり，近年，技術的負債に関する研究は増加傾向にある [14]．

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文書化し明示した特殊な事象が存在する．このようなケースを Self-Admitted Technical

[1] 和歌山大学
Wakayama University
[a)] kimura.yuta.1@g.wakayama-u.jp
[b)] masao@wakayama-u.ac.jp

Debt（SATD）という．SATD には 2 種類存在し，ソースコード中にコメントを用いた SATD と課題票（Issue）を用いた SATD がある．本論文では，ソースコードコメントを用いた SATD を SATD-Comment，Issue を用いた SATD を SATD-Issue と呼称する．SATD-Comment は，Potdar と Shihab [19] によって初めて導入され，4 つの OSS プロジェクトを対象に調査をおこない，SATD-Comment が広く見られる事象であることが明らかにされている．これは，これまで静的コード解析によってコードの情報を分析し技術的負債を検出していたが，テキスト情報を用いることでも技術的負債が検出できることを示している．そのため，SATD-Comment に関する研究は盛んにおこなわれている．SATD-Issue は，Xavier ら [26] によって初めて導入され，いくつかのオープンソースソフトウェア（OSS）プロジェクトで意識的に技術的負債を管理していることを明らかにしている．しかし，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．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 SATD-Issue を扱っていることがわかった．そのうち，open，closed 合わせて 100 件以上の SATD-Issue を保有している OSS プロジェクトは 2 件（0.2 %）しか存在しない．

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 の問題を解決できると考える．技術的負債に関連する Issue が抽出可能になることで，さまざまな研究の発展が見込める．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，前節でも述べたように，Issue は解決時の変更を紐づけることができるため，開発者が問題と捉える技術的負債の実態を明らかにできる可能性がある．実態を明らかにすることで，例えば，これまでに研究されたきた技術的負債との比較調査で開発者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，開発者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．

本論文では，SATD-Issue の問題解決の第一歩として，SATD-Issue の自動分類モデルの構築をおこなう．本論文では主に 2 つの調査をおこなう．まず，(1) SATD-Issue を決定づける特徴の分析をおこない，(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おこなう．(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おこなう．(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精度の評価をおこなう．

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た．
- SATD-Issue の自動分類モデルを構築し，最も分類精度が良いモデルでは Precision が 0.995，Recall が 0.903，F1 値が 0.947 となった．
- 分類精度の向上に最も寄与している特徴は報告者の特徴であり，最も寄与していない特徴はソースコードの特徴であることがわかった．

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-Issueno 特徴分析をおこなう．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 [26] 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．そのため，報告された当初にタグけられたラベルが取り除かれたり，新たなラベルがあとからタグ付けられたりする．よって，本研究では，問題内容が確定しているクローズ（解決）している，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 とみなす．

2.2 データの収集

本論文では，[26] 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．

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おこなう．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．

表 1: 対象リポジトリと Issue 数

リポジトリ	ラベル	SATD-Issue	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

3.1 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 [19]．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．本研究では，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おこな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 [26]．くわえて，3.1 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 [17,23] 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 を解決するためにおこな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．例えば，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 [3,9]．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．本研究では，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おこなう．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捉える．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．

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 [21,25] ではテキストをベースとした分類モデルが提案されている．本研究でも同様に，テキストをベースとした分類モデルの構築をおこなう．今回は，テキスト以外に分類精度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おこなう．

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おこなう．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 に示している．有意差が見られた特徴に

[*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

はアスタリスクがつけられており，効果量が Small 以上の場合には正負の強さを示している．3.5.1 で述べたように，今回の分析ではテキスト以外に分類精度の向上に影響しそうな特徴を明らかにするため，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．

4. 分類モデル

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おこなう．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 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[*2] 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がさらに分割できるかどうかチェックし繰り返し分割をおこなう．

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，本研究でも “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おこない，固有名詞と Out-of-Vocabulary [*3]（OOV）と判定された単語の置換をおこなう．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，全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）をおこない表記を統一する．

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．

[*2] https://spacy.io/usage/spacy-101#annotations-token
[*3] spaCy のコーパスに含まれていない単語（ユーザ独自の造語やスラングなど）のこと

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 [12]，ロジスティック回帰 [2]，Support Vector Machine [5] 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 は正則化項を L2 ノルム，反復回数を 2000 とし，ランダムシードは全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おこなう．具体的には，Lasso 回帰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おこなう．

5. 評価実験

前節で構築した分類モデルの評価実験をおこなう．構築する分類モデルの分類精度を明らかにすることで，プロジェクトごとに SATD-Issue を導入する基準がどの程度定まっているのかを評価する．分類精度が高いほどプロジェクトにおける SATD-Issue の導入基準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の構築し分類精度の評価をおこなう．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 回おこなう計 100 回の平均分類精度を評価する．

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

表 2: 特徴と分析結果

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

慮する必要がある．本研究では，多重共線性の指標として Variance Inflation Factor（VIF）を利用する．算出された VIF がもっとも大きい値となる特徴を削除し，全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おこなう．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）

リポジトリ	SATD-Issue	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 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 に示している．

表 4: 分類精度

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

5.3 評価指標

分類精度の評価指標には，Precision と Recall，F1 値を用いる．SATD-Issue を SATD-Issue と予測したものを真陽性（TP：True Positive），そのほかの Issue を SATD-Issue と予測したものを偽陽性（FP：False Positive），SATD-Issue をそのほかの Issue と予測したものを偽陰性（FN：False Negative）とすると，Precision と 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 の値を取り，予測した Issue 全てが SATD-Issue だった場合に Precision は 1 となり，全ての 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．

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

5.4 実験結果

各分類モデルの分類精度の評価結果を表 5 に示している．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）でおこなう．そのため，各プロジェクトにおいて Recall が最も高くなったモデルの値を太字で強調している．

microsoft/vscode：TF-IDF と SVM を用いたモデルが最も分類精度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．

influxdata/influxdb：BoW と SVM を用いたモデルが最も分類精度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

saleor/saleor：BoW と SVM を用いたモデルが最も分類精度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．

nextcloud/server：TF-IDF と SVM を用いたモデルが最も分類精度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．

6. 精度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，分類精度の向上に寄与している特徴を明らかにするためにアブレーションスタディをおこなう．アブレーションスタディでは，評価実験で最も良い分類精度であった vscode の分類モデル（TF-IDF と SVM を用いたモデル）を利用する．テキストのみを利用した分類モデルをベースラインとし，報告者，プロセス，ソースコードそれぞれの特徴を足し引きし構築した分類モデルと比較する．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 回おこない，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精度からの向上率の算出とマンホイットニーの U 検定を用いた分類精度の有意差検定（$p < 0.01$）をおこなう．

アブレーションスタディの結果を表 5 に示している．まず，ベースラインであるテキストのみを利用した分類モデルの分類精度に注目すると，Precision が 0.852，Recall が 0.462 と比較的高い精度で分類できていることがわかる．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 つを加えた分類モデルに注目すると，報告者とプロセスを加えた分類モデルでベースラインの分類精度において有意差が見られている．このことから，ソースコードの特徴が分類精度の向上に寄与しないことがわかった．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が最も精度の向上率が低いことがわかる．このことから，報告者の特徴が分類精度の向上に最も寄与している特徴であることがわかった．

7. 妥当性への脅威

7.1 内的妥当性

SATD-Issue の定義：本研究では，SATD-Issue 特定の

表 5: アブレーションスタディの結果（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	(+44.2%)	*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
*：$p < 0.01$

ために Issue のラベル機能を用いている．特定には “deb” や “technical deb” 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：本研究では，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おこなうことで本研究の結果が異なる可能性がある．

ソースコードの情報取得と利用した特徴：本研究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おこな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

本研究では，4 つの OSS プロジェクトを対象に調査をおこなっている．対象としてるプロジェクトが少ないため，本研究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盛んにおこなわれている．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 [3,13,27]．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 [4,18]．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 [1,10] も多く提案されている．

8.2 SATD-Comment 検出に関連する研究

Potdar と Shihab [19] 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．[19] 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 [7] は，Contextualized Vocabulary Model (CVM) の提案をおこなっている．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 [11,16] 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 [8,17,20]．

9. おわりに

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 の問題解決の第一歩として，SATD-Issue の自動分類モデルの構築をおこなった．主に 2 つの調査をおこない，(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おこなった．評価実験では，分類精度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精度が良く 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おこない分類精度の向上に寄与している特徴の調査をおこなったところ，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わかった．

今後は，テキストの利用方法に関するブラッシュアップなどをおこない分類モデルの実用化を図る．その後，SATD-Issue の自動分類モデルを用いて，SATD-Issue を扱っていないプロジェクトへの適用をおこないたいと考えている

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, *Proceedings of the 26th International Symposium on Software Reliability Engineering*, ISSRE, 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] Bohnet, J. and Döllner, J.: Monitoring Code Quality and Development Activity by Software Maps, *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD, pp. 9–16 (2011).

[4] Brondum, J. and Zhu, L.: Visualising Architectural Dependencies, *Proceedings of the 3rd International Workshop on Managing Technical Debt*, MTD, pp. 7–14 (2012).

[5] Cortes, C. and Vapnik, V.: Support-Vector Networks, *Machine Learning*, Vol. 20, pp. 273–297 (1995).

[6] Cunningham, W.: The WyCash Portfolio Management System, *SIGPLAN OOPS Mess.*, Vol. 4, No. 2, pp. 29–30 (1992).

[7] 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, *Proceedings of the18th International Conference on Enterprise Information Systems*, ICEIS, pp. 284–309 (2016).

[8] 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).

[9] Fontana, F. A., Ferme, V. and Spinelli, S.: Investigating the Impact of Code Smells Debt on Quality Code Evaluation, *Proceedings of the 3rd International Workshop on Managing Technical Debt*, MTD, pp. 15–22 (2012).

[10] Fontana, F. A., Zanoni, M., Marino, A. and Mäntylä, M. V.: Code Smell Detection: Towards a Machine Learning-Based Approach, *Proceedings of the 29th International Conference on Software Maintenance*, ICSM, pp. 396–399 (2013).

[11] 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).

[12] Ho, T. K.: Random Decision Forests, *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, ICDAR, pp. 278–282 (1995).

[13] Letouzey, J.-L. and Ilkiewicz, M.: Managing Technical Debt with the SQALE Method, *IEEE Software*, Vol. 29, No. 6, pp. 44–51 (2012).

[14] 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).

[15] 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).

[16] 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, *Proceedings of the 40th International Conference on Software Engineering: Companion*, ICSE-Companion, pp. 9–12 (2018).

[17] Maldonado, E. d. S., Shihab, E. and Tsantalis, N.: Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt, *IEEE Transactions on Software Engineering*, Vol. 43, No. 11, pp. 1044–1062 (2017).

[18] Mo, R., Garcia, J., Cai, Y. and Medvidovic, N.: Mapping Architectural Decay Instances to Dependency Models, *Proceedings of the 4th International Workshop on Managing Technical Debt*, MTD, pp. 39–46 (2013).

[19] Potdar, A. and Shihab, E.: An Exploratory Study on Self-Admitted Technical Debt, *Proceedings of the 30th International Conference on Software Maintenance and Evolution*, ICSME, pp. 91–100 (2014).

[20] 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 Transaction Software Engineering and Methodology*, Vol. 28, No. 3, pp. 1–45 (2019).

[21] Runeson, P., Alexandersson, M. and Nyholm, O.: Detection of Duplicate Defect Reports Using Natural Language Processing, *Proceedings of the 29th International Conference on Software Engineering*, ICSE, pp. 499–510 (2007).

[22] 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, *Proceedings of the 3rd International Workshop on Managing Technical Debt*, MTD, pp. 45–48 (2012).

[23] Tan, J., Feitosa, D. and Avgeriou, P.: An Empirical Study on Self-Fixed Technical Debt, *Proceedings of the 3rd International Conference on Technical Debt*, TechDebt, pp. 11–20 (2020).

[24] Tom, E., Aurum, A. and Vidgen, R.: An Exploration of Technical Debt, *Journal of Systems and Software*, Vol. 86, No. 6, pp. 1498–1516 (2013).

[25] Wang, X., Zhang, L., Xie, T., Anvik, J. and Sun, J.: An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information, *Proceedings of the 30th International Conference on Software Engineering*, ICSE, pp. 461–470 (2008).

[26] Xavier, L., Ferreira, F., Brito, R. and Valente, M. T.: Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems, *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR, pp. 137–146 (2020).

[27] 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).