Vol.2019-SE-201 No.15
2019/3/8

Self-Admitted Technical Debtの存在期間・除去人物についての追実験

西川 諒真[1,a)]　西中 隆志郎[1,b)]　亀井 靖高[1,c)]　佐藤 亮介[1,d)]　鵜林 尚靖[1,e)]

概要：Self-Admitted Technical Debt（SATD）とは開発上での設計における理想状態からの乖離を示す技術的負債の一種であり，ソースコード上へ開発者がコメントを用いて自ずから言及した技術的負債のことを指す．SATD について研究することはプロジェクトに存在する改善点を容易に知ることに繋がり，プロジェクトの円滑な開発に大きく貢献する．しかし，SATD に関する研究は未だ不十分であり，実証研究が少ない．そこで，本研究では既存研究の追実験に加え新たなデータセットを使用することで，研究結果の一般性や新たな知見の獲得を試みた．GitHub から 10 の Java プロジェクトを取得し，1) どれほどの SATD が除去されているか，2) SATD が混入してから除去されるまでどれほどの期間があるか，3) 誰が SATD の除去を行っているかについて調査した．結果として，除去された SATD の割合が 23.2〜100.0% で中央値 43.4%であること，SATD の混入から除去まで 212.9〜4822.7 日の期間が存在すること，除去された SATD のうち 0.0〜95.7%，中央値 48.5%がその SATD の混入を行った開発者自身により除去されていることが分かった．

1. はじめに

技術的負債とは開発上における設計状態の理想状態からの乖離を示す現象である．技術的負債を検出するための方法として，コードの不吉な臭いやコーディングスタイル違反といった指標が用いられてきた [2][5]．その中でもコメントを用いて技術的負債を探る方法が示されており，SATD はコメントを用いてソースコード上へ開発者自身によって言及が行われた技術的負債のことを指す [9]．

SATD について知ることは，その負債への扱い方を知ることに繋がり，プロジェクトの開発を円滑に進めることに貢献する．SATD を用いてプログラムの状態の不備を示す開発者は多く，例えば，Vassallao ら [10] は金融機関の開発者 152 人を対象に調査を行い，88 %の回答者がソースコード内にコメントを用いてプログラム実行のクオリティの低さ（すなわち技術的負債）について言及していることを示した．そのため，コメントベースである SATD の研究は技術的負債の処理に有用であると考えられる．

SATD に関する実証的研究は十分でなく，特に除去に着目した研究には課題がある [3]．Maldonado ら [3] は，一部の SATD がプロジェクト内へ混入後に除去されず悪影響を与え続けていることから，新たに SATD の除去に着目した研究を行った．その研究では Camel, Gerrit, Hadoop, Log4j, Tomcat の 5 つのプロジェクトに対してデータセットを取得し，どれほどの割合の SATD が除去されているか，誰が SATD を除去することが多いのか，SATD の除去までにかかる期間を示し，開発者へのオンライン調査で得た回答から SATD の除去に至る理由の分類を行った．しかし，この調査で得られた結果にはプロジェクトによる差が大きく，対象としたプロジェクト数の少なさについての課題点が存在する．

そこで本研究では，新たなデータセットを使用して Maldonado らの調査の追実験を行うことで，SATD に関する実証的な分析結果の一般性の向上を試みる．本稿では実験の第一段階として定量的な実験のみに限定して追実験を行った．1000 以上のコミット数を持つ 10 の Java オープンソースプロジェクトから 853 の SATD を抽出し，1) どれほどの SATD が除去されているか (RQ1)，2) SATD が混入してから除去されるまでどれほどの期間があるか (RQ2)，3) 誰が SATD の除去を行っているか (RQ3) について調査を行った．

以降，第 2 章では関連研究について述べる．第 3 章では本研究の実験計画，使用したデータセットとその選択基準

[1] 九州大学
Kyushu University
[a)] nishikawa@posl.ait.kyushu-u.ac.jp
[b)] nishinaka@posl.ait.kyushu-u.ac.jp
[c)] kamei@ait.kyushu-u.ac.jp
[d)] sato@ait.kyushu-u.ac.jp
[e)] ubayashi@ait.kyushu-u.ac.jp

について説明を行う．第 4 章では調査課題と各調査課題のアプローチ，そして得られた結果を述べる．第 5 章では妥当性への脅威について述べる．最後に第 6 章では本研究のまとめと今後の展開を述べる．

2. 関連研究

Bavota ら [1] は，混入した SATD の時間経過による増減についての研究が少なかったことに着目し，研究を行った．ソフトウェアプロジェクトから約 300 のコメントを抽出し，混入した SATD の内容の分類に加え，頻出する SATD の種類の傾向，プロジェクトごとに平均 50 程度の SATD が存在すること，SATD の修正が行われなければ SATD が増幅し，修正後も残りやすいことを示した．これに対し，本研究は SATD の除去にかかる期間と，除去を誰が行うかについてを調査する．将来的には除去についての分類も行う予定である．

亀井ら [6] は SATD の除去におけるコストについて広く研究されていなかったことから，SATD の利子，すなわち混入した SATD の大きさが混入後にどう変わるかについての研究を行った．研究の結果，コードの行数，論理入力数のそれぞれの観点において，SATD の内 40 %超に正の利子が存在する，つまり SATD が増幅するということを示した．この研究では利子の計算の際，プロジェクト内に SATD が混入したときと除去されたときの二つのリビジョンを利用し，それぞれの SATD の大きさの差から除去の重要性を示している．これに対し，本研究では SATD の混入と除去のリビジョン間にある時間差の観点から，除去についての調査を行う．

Potdar ら [9] は急いで行われた修正や一時的な修正による誤りの影響についての研究が過去に十分行われていなかったという観点から，Self-Admitted Technical Debt という概念を導入し，約 10 万のコメントを分析して SATD に該当するコメントのパターンを 62 種類に分類した．加えて，4 つのプロジェクトに対して調査を行い，2.4〜31.0% のファイルに SATD が含まれていたこと，経験の多い開発者ほど SATD の混入を行うことが多いこと，リリースまでの時間やコードの複雑さが SATD への強い相関がなく，リリース後にも SATD の除去は行われていることを示した．この研究ではパターンマッチで SATD の検出を行っているが，本研究では，テキストマイニングを用いたツールにより SATD の検出をより高い精度で行う点で新規性を持つ．

Maldonado ら [4] は，過去の SATD の検出方法の多くが手動に依存していることにより，検出に手間がかかり，なおかつ人力によるバイアスがかかっていたことから，自然言語処理（NLP）を利用した SATD の自動検出の手法を提案した．加えて，実際に適用して SATD の検出を行なった結果，従来より簡単に，かつ高い精度で SATD を検出できたことを示した．NLP を用いた SATD の自動検出は先述の Portdar らの手法に比べ精度が高い．Maldonado らは SATD の除去に関する実験において，この手法を用いて SATD の検出を行っているが，本研究で用いる SATD の検出ツールはそれよりもさらに精度の高いものとなっている．

Wehaibi ら [11] は，SATD がソフトウェアの品質に与える影響に着目した実証研究が過去に無かったことに着目し，5 つのプロジェクトをデータセットに用いて SATD とソフトウェアのクオリティとの関連性を調べた．SATD を含むファイルは SATD を含まないファイルよりも欠陥が多いのか，SATD に関連する変更はその後欠陥を誘引しやすいのか，SATD に関連する変更はそうでないものより難しいのかについて調査し，SATD と欠陥の間に明確な傾向は見られないものの，SATD が混入したファイルには混入前と比べて欠陥の修正が起こりやすいこと，SATD に関する変更は SATD に関係しない変更よりも欠陥を誘引しにくくすること，SATD に関する変更は SATD に関係しない変更よりも難しいことを示した．この研究では SATD による欠陥の発生に着目して調査が行われているのに対し，本研究では SATD そのものに着目して調査を行う．

3. 実験設計

本章では使用したプロジェクトについて，およびその詳細な選択条件を説明する．

3.1 データセット

本研究では Maldonado らの使用したプロジェクトとは異なる，新規のプロジェクトをランダムに取得する．新しく取得するデータセットに使用するプロジェクトの選択基準については，山下ら [12] の手法を参考にした．

プロジェクトの取得には GitHub の API を利用し，1) SATD の検出に利用する SATD Detector[8] のサポート対象である Java 言語を用いていること，2) オープンソースであること，3) フォークされたリポジトリではないこと，4) 開発者数が 10 人以上であること，5) フォーク数が 10 以上であること，6) Java ファイルサイズが 500KB 以上であること，7) Java ファイルがプロジェクト全体の中で半分以上の容量を占めること，8) 1000 以上のコミット数があることから，pwm, molgenis, maxwell, cloudfly, askyblock, apg, eclipse.platform.text, hibernate-search, stripe-java, Perl5-IDEA の 10 プロジェクトを選択した．

本研究では，各プロジェクトにおいて，それぞれ git log コマンドによって新しい順に取得できる全リビジョンの中から中央に位置する 1 リビジョンを使用した．各プロジェクトの詳細を表 1 に示す．表中において，SLOC とはソースコードのうち，空白のみの行やコメントのみの行を除外した行数のことを指す．

表 1　データセット

Project	使用したリビジョン	# Java files	Java SLOC	# commits	# contributors	# SATD comments	概要
pwm[*1]	9c754d3	753	108,922	2,417	27	10	パスワード管理アプリケーション
molgenis[*2]	fbe452d	2,036	167,991	21,921	36	216	科学データ用アプリケーション
maxwell[*3]	28133e1	212	16,823	2,724	71	8	MySQL から JSON への書き込み用アプリケーション
cloudify[*4]	756ee02	1,034	78,153	11,048	22	112	クラウド対応プラットフォーム
askyblock[*5]	43c9a2b	130	30,727	1,574	33	30	ゲーム用プラグイン
apg[*6]	18430be	273	41,992	4,367	57	86	Android 用暗号化ソフトウェア
eclipse.platform.text[*7]	d52f0a1	1,218	141,036	6,796	57	93	ユーザーインタフェース用プロジェクト
hibernate-search[*8]	3f2ecf2	4,168	261,103	7,442	48	151	Java オブジェクト用の全文検索モジュール
stripe-java[*9]	4abe9ca	333	16,042	1,368	119	7	Stripe API 用 Java ライブラリ
Perl5-IDEA[*10]	f2c2d42	1,553	78,329	4,111	11	140	IntelliJ IDEA 用プラグイン

3.2　選択条件

本項ではプロジェクトの選択条件の具体的な内容とその理由を説明する．

3.2.1　フォーク

GitHub 上には，メインリポジトリとフォークリポジトリが存在する．フォークとはメインリポジトリのコピーを取ることであり，フォークリポジトリはプルリクエストによりメインリポジトリに変更を反映させることができる．受け入れられたプルリクエストは全てメインリポジトリへと保存されるため，フォーク先のリポジトリは除外する．また，ランダムにリポジトリを選択する際，フォーク数が多いリポジトリほど選択されやすくなるため，確率の平等性を保つ意味においてもフォークリポジトリは除外する．

3.2.2　開発規模

開発者数が少ないと SATD の除去人物を求める際に偏りが出る可能性がある．極端な例を挙げると，開発者数が 1 人の場合は SATD をどう扱うかに関係なく，SATD の除去人物がその開発者 1 人に必ず定まってしまう．このような規模の小ささによる結果の偏りを避けるため，開発者が 10 人未満の場合は除外する． 同様に，小規模のリポジトリの選択を避けるため，フォーク数が 10 未満のものも除外する．また，コミットが頻繁に行われないプロジェクトの場合は SATD の混入時期，除去時期を正確に測れない可能性があるため，コミット数が 1000 以下のリポジトリは除外する．

3.2.3　Java

SATD の検出に利用する SATD Detector は Java 言語のみをサポート対象としているため，Java ファイルが少ないリポジトリ，具体的にはリポジトリ内の Java ファイルの合計容量が 500KB 未満のものを除外する．また，Java が主な言語として使用されているリポジトリのみを取得するために，Java ファイルの容量の占める割合が全体のうち 50%未満であるリポジトリを除外する．

3.2.4　SATD Detector の実行

SATD Detector は，テキストマイニングをベースとした手法で SATD の自動検出を行う，Java 用の Eclipse プラグインである．SATD の検出を行いたいプロジェクトを Eclipse のワークスペース内にインポートすることで，自動的に SATD の検出を行う．検出した SATD は Eclipse 内のタスク欄に出力され，SATD と，その SATD が存在するパス，ファイル名，ソースコード内での行数を表示する．本研究ではこのタスク欄への出力を利用して実験を行った．

4.　実験方法と結果

本研究の目標は Maldonado らの研究の追実験を行い，結果を拡大することである．そのため，本研究では以下の調査を行う．

RQ1：どれほどの SATD が除去されているか

RQ2：SATD が混入してから除去されるまでどれほどの期間があるか

RQ3：誰が SATD の除去を行っているか

Maldonado らはこれに加えて SATD の除去を行う理由についてのオンライン調査を行っているが，本研究では実験の第一段階として定量的な実験のみを行い，オンライン調査は行わない．

4.1　(RQ1) どれほどの SATD が除去されているか

動機：一つ目の RQ は SATD の除去の割合についてである．SATD に限らず，過去の研究では，技術的負債は発生が避けられないものであり，プロジェクトに対して悪影響を与えているものであるということが示されている [7]．このことから，開発者は基本的に SATD を取り除くであろうと推測できる．開発者が SATD にどのように対処しているかを理解するために，まずは SATD がどれほど除去されているかについて調査を行う．

アプローチ：図 1 にアプローチの概略を示す．ただし RQ1 においては図中に記載されている SATD の混入時期の特定

*1 https://github.com/pwm-project/pwm
*2 https://github.com/molgenis/molgenis
*3 https://github.com/zendesk/maxwell
*4 https://github.com/CloudifySource/cloudify
*5 https://github.com/tastybento/askyblock
*6 https://github.com/thialfihar/apg
*7 https://github.com/eclipse/eclipse.platform.text
*8 https://github.com/hibernate/hibernate-search
*9 https://github.com/stripe/stripe-java
*10 https://github.com/Camelcade/Perl5-IDEA

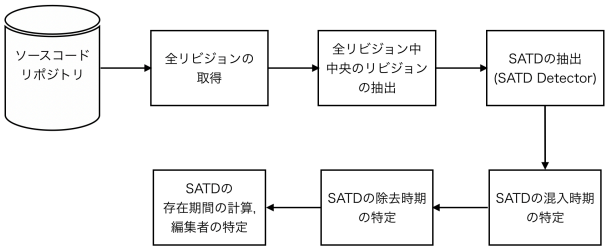

図 1 アプローチの概略

は必要ない．

はじめにプロジェクトから全リビジョンのうち中央に存在するリビジョンを抽出した後，まずリポジトリから SATD Detector を用いて SATD を検出する．SATD の検出後，git の git blame –reverse コマンドを利用することで，どの変更まで SATD が存在したか，およびその時刻を抽出することができる．さらに git log コマンドを使用してその SATD が含まれるファイルの変更履歴を取得し，git blame –reverse で取得した時刻より後に変更が存在した場合，SATD の除去があったと判断した．

結果 :結果を表 2 に示す．除去された SATD の割合の中央値は 43.4%となった．Maldonado らの研究結果では除去された SATD の割合の中央値は 76.7%であり，比較すると低い値となり，SATD のうち，およそ半数ほどの SATD が除去されずに残っているという結果となった．

検出した SATD の数の大小に関わらず除去の割合が高いものから低いものまであるため，プロジェクト毎による依存が強いと考えられる．そこで，プロジェクトの何が SATD の除去の割合に影響しているかを調べるため，各プロジェクトの SATD の除去の割合と，Java ファイル数，SLOC，コミット数，開発者数，SATD の数との相関係数，およびそれに対する無相関検定の結果を求めた．表 3 にその結果を示す．最も大きい相関が見られたのは SLOC で，負の相関となった．

表 3 中の全項目で p 値が 0.05 以上となったため無相関検定においては有意性を認められなかったものの，SLOC が大きくなるほど，除去される SATD の割合が低くなる可能性が考えられる．なお，SLOC と SATD の除去率の相関関係の有意性を示すには，相関係数-0.519 に対してプロジェクト数が 15 以上必要となる．SLOC が SATD の除去の割合に影響を与える要因として考えられるものには，コードの行数が大きくなることによる，プロジェクト全体の管理の難化が存在することが理由の 1 つとして考えられる．

> 中央値で 43.4%，平均で 58.9%の割合で SATD の除去が行われていた．

表 2 RQ1 結果

Project	# Identified	# Removed	% Removed	% Remaining
pwm	10	4	40.0	60.0
molgenis	216	88	40.7	59.3
maxwell	8	3	37.5	62.5
cloudify	112	112	100.0	0.0
askyblock	30	15	50.0	50.0
apg	86	51	59.3	40.7
eclipse.platform.text	93	36	38.7	61.3
hibernate-search	151	35	23.2	76.8
stripe-java	7	7	100.0	0.0
Perl5-IDEA	140	139	99.3	0.7
Average	-	-	58.9	41.1
Median	-	-	43.4	56.6
Maldonado らの中央値	-	-	76.7	23.3

表 3 SATD の除去率と各項目の相関係数

	# Java files	Java SLOC	# commits
相関係数	-0.338	-0.519	-0.144
p 値	0.339	0.124	0.692
	# contributors	# SATD comments	
相関係数	0.053	-0.063	
p 値	0.885	0.861	

4.2 (RQ2)SATD が混入してから除去されるまでどれほどの期間があるか

動機 :二つ目の RQ は SATD の存在期間についてである．本 RQ に対する答えは，SATD が一般的にプロジェクトにどれほどの期間存在するかを知ることに繋がる．SATD は放置しているとその大きさが増大することが示されている [6] ため，SATD は早く除去されることが望ましいが，開発上すぐに除去できない場合も存在する．それゆえ，SATD の一般的な存在期間を知ることは除去を行うべき時期の指標を立てる上で役立つと考えられる．

アプローチ :RQ1 とほぼ同じアプローチを使用する．RQ1 にて SATD を検出した後，git の git blame コマンドを使い，SATD の存在する行に対して最後の変更が行われたリビジョンを取得することにより SATD が初めて記載された時期，すなわち SATD の混入時期を特定する．その後，RQ1 と同様に git blame –reverse，git log コマンドを使い，SATD の存在する行に変更が行われたリビジョンを SATD の除去時期とした後，除去時期と混入時期の時間差を取ることで SATD の存在期間を計算する．なお，除去が行われていない SATD については，リポジトリを取得した時刻（2019 年 1 月 27 日 15 時 40 分）をその SATD が除去された時刻として扱う．

結果 :結果を表 4 に示す．中央値は小さいもので 212.9 日，大きいもので 4822.7 日となっている．Maldonado らの研究結果はプロジェクトごとの中央値で 18.2〜172.8 日であるため，比較すると極めて大きな値となった．

これには，SATD の検出方法によるデータセットの偏りが考えられる．現在の手法ではプロジェクト中の 1 つのリ

表 4　RQ2 結果

Project	Min.	1st quartile	Median	3rd quartile	Max.
pwm	341.1	406.1	582.0	609.0	1276.6
molgenis	6.2	245.9	580.9	1101.6	1804.8
maxwell	114.0	227.9	369.0	975.5	1339.7
cloudfly	22.8	160.1	212.9	256.8	484.3
askyblock	66.1	242.1	761.4	896.8	1016.78
apg	2.9	128.8	217.7	401.6	981.9
eclipse.platform.text	45.3	958.1	4822.7	5241.2	5810.0
hibernate-search	196.8	1600.8	2151.1	3028.5	4038.6
stripe-java	449.0	674.5	890.5	998.9	1388.5
Perl5-IDEA	85.6	365.7	446.9	660.2	1095.4
Average	133.0	501.0	1103.5	1412.0	1923.6
Median	75.9	305.8	581.5	936.2	1308.1
Maldonado らの中央値	-	-	159.0	-	-

表 5　RQ3 結果

Project	# Removed	# Self-removed	% Self-removed
pwm	4	3	75.0
molgenis	88	5	5.7
maxwell	3	2	66.7
cloudify	112	3	2.7
askyblock	15	13	86.7
apg	51	31	60.8
eclipse.platform.text	36	13	36.1
hibernate-search	35	7	20.0
stripe-java	7	0	0.0
Perl5-IDEA	139	133	95.7
Average	-	-	44.9
Median	-	-	48.5
Maldonado らの中央値	-	-	61.0

ビジョンの SATD を検出し，その混入時期，除去時期を特定することで調査を行なっているが，そのリビジョン内に無い SATD は検出できない．そのため，混入から除去までの時間が短い SATD は検出しづらく，逆に混入から除去までの時間が長い SATD は検出しやすい，という結果の偏りが生じている可能性がある．

SATD が混入してから除去されるまで，中央値で 212.9〜4822.7 日の期間があった．

4.3　(RQ3) 誰が SATD の除去を行っているか

動機 :三つ目の RQ は SATD の除去を行う人物についてである．SATD は開発者自身によって混入が行われるため，直感的には他の人物に比べ，混入者自身の方がその SATD への理解が深く，除去を行うのが簡単で，一般的であると推測できる．しかし，実際に混入者自身で SATD の除去が主に行われているかは定かではない．本 RQ の結果は，SATD をどのように扱うべきかを知ることに役立つ．例えば，混入者以外によって SATD が除去されることが多いならば SATD はプロジェクト全体で共有，把握される必要があるものといえ，そうでないならば混入者各自によって処理されるものであるといえる．

アプローチ :SATD の混入者・除去者を特定する．SATD の混入，除去が行われたリビジョンを特定する際，そのリビジョンにおいて git の管理システムに登録されている編集者を混入者，除去者として扱う．なお，除去が行われていない SATD は使用しない．SATD の混入，除去の行われたリビジョンを特定する方法については RQ2 と同じアプローチを使用する．

表 6　混入者自身による SATD の除去率と各項目の相関係数

	# Java files	Java SLOC	# commits
相関係数	-0.340	-0.343	-0.564
p 値	0.336	0.332	0.089
	# contributors	# SATD comments	% removed SATD
相関係数	-0.413	-0.337	-0.118
p 値	0.235	0.341	0.746

結果 :結果を表 5 に示す．混入者自身による SATD の除去の割合は中央値で 48.5%となった．Maldonado らの研究結果は中央値で 61.0%であるため，比較すると低い値となった．

また，プロジェクト毎の結果の差が大きかったため，結果への影響要因を調べるために RQ1 と同様にプロジェクト毎の混入者自身による SATD の割合と各項目の相関係数，およびそれに対する無相関検定の結果を計算した．結果を表 6 に示す．結果，コミット数において最も大きい負の相関が見られ，SATD の除去率と混入者自身による SATD の除去率の相関以外において全体的に小さい負の相関が見られた．無相関検定による有意性は認められなかったものの，コミット数，もしくはその他の開発規模に関するいずれかの要因に応じて混入者以外が SATD の除去を行う割合が増えていると考えられる．なお，コミット数と SATD の除去率の相関関係の有意性を示すには，相関係数-0.564 に対してプロジェクト数が 13 以上必要となる．現段階ではどの要素が影響要因となっているかの判断が難しいため詳細な推測は困難であるが，具体的にどういった人物が主に除去を行っているかが分かれば考察の手掛かりになる可能性がある．

中央値で 48.5%，平均で 44.9%の割合の SATD が，混入者自身によって除去されていた．

4.4　従来研究との比較

本節では Maldonado らの研究との比較を行う．表 7 に Maldonado らの使用したデータセットを示す．Maldonado らはプロジェクトの取得後，各プロジェクトの全リビジョ

*11 https://github.com/apache/camel
*12 https://github.com/gerrit-review/gerrit.git
*13 https://github.com/apache/hadoop
*14 https://github.com/apache/log4j
*15 https://github.com/apache/tomcat

表 7　従来研究でのデータセット

Project	# Java files	Java SLOC	# file versions	# contributors	# SATD comments	概要
Camel[*11]	15,091	800,488	254,920	289	4,331	統合フレームワーク
Gerrit[*12]	3,059	222,476	53,298	270	271	コードレビュー，プロジェクト管理ツール
Hadoop[*13]	8,466	996,877	79,232	160	1,164	分散処理プラットフォーム
Log4j[*14]	1,112	30,287	12,609	35	135	ロギングユーティリティ
Tomcat[*15]	3,187	297,828	46,716	32	1,317	Java servlet 用サーブレットコンテナ

表 8　従来研究との比較

RQ	本研究の結果	従来研究の結果
(RQ1) SATD の除去の割合（中央値）	43.4%	76.7%
(RQ2) SATD の存在期間（中央値）	212.9〜4822.7 日	18.2〜172.8 日
(RQ3) 混入者自身による SATD の除去の割合（中央値）	48.5%	61.0%

ンを抽出している．

各 RQ ごとの比較を表 8 に示す．本研究の結果は，Maldonado らの研究結果と比較すると，1) SATD の除去の割合が低い，2) SATD の混入から除去までの期間が極めて長い，3) 混入者自身による SATD の除去の割合が低い，という結果となった．

5. 妥当性への脅威

本章では，妥当性への脅威について議論する．

5.1 内的妥当性

内的妥当性では，調査結果へ影響を与えた可能性がある要因について議論する．一つ目に，SATD の検出に使用した SATD Detector が考えられる．SATD Detector は機械学習を用いて作られており，その F 値は平均 0.737. precision は平均 0.756, recall は平均 0.733 となっている．このため，SATD の検出の精度がツールの精度に依存する．

二つ目に，git のコマンドを利用した SATD の混入・除去時期の特定方法が考えられる．git blame，git blame –reverse は行単位でソースコードの変更を見るコマンドである．同じ行ならば SATD 以外の部分が変更されていても検知してしまうため，必ずしも正しい混入時期・除去時期を取れていない可能性がある．また，SATD 自体に変更があったとしても，小さな誤字の修正などといった除去と関係ない要因での変更であることもあり得る．対応としては，手法そのものを変えるか，コマンドの使用後，SATD の存在する行の内容を読み込み，SATD 自体に変化があったかどうかを調べるといった処理を加えることが考えられる．その場合，単純に変化があったかではなく，どの程度変化したかの閾値を設ける必要がある．

三つ目に，RQ2 にて述べた，1 つのリビジョンのみから SATD を検出していることによるデータセットの偏りも考えられる．この問題を解決するためには別のリビジョンを利用してより多くの SATD を検出する必要があるが，現手法で全てのリビジョンを網羅するのは現実的な手段ではないため，アプローチの改善を試みている．

5.2 外的妥当性

外的妥当性では調査結果の一般化において影響を与える事柄について議論する．その要因として，対象としたプロジェクトの条件が考えられる．本研究ではいずれも GitHub 中に存在するもののみ，Java のみを対象としており，他の言語，および他のオープンソースや商用のシステムにおいての一般化はできていない可能性があるが，10 のオープンソースプロジェクトを対象に調査を行った．

6. まとめと今後の展開

SATD とは, 開発者が自ずからソースコード上で言及を行なった技術的負債のことである．本研究では，Maldonado らの SATD の除去に関する実証研究の追実験を行い，どれほどの割合の SATD が除去されているか，SATD が混入してから除去されるまでどれほどの期間があるか，誰が SATD の除去を行っているかを調査した．その結果，以下の結果が得られた．

- プロジェクト別で 23.2〜100.0%，中央値で 58.9%の SATD が除去されており．その除去率と SLOC の間に弱い負の相関が見られた．
- SATD が混入してから除去されるまで，プロジェクト毎の中央値で 212.9〜4822.7 日存在し続けていた．
- 除去された SATD のうち，プロジェクト別で 0.0〜95.7%，中央値で 48.5%がその SATD を混入させた開発者自身によって除去されており，混入者自身による SATD の除去の割合とコミット数との間に弱い負の相関が見られた．

今後の展開としては，アプローチの改善に加え，追加のデータセットの取得や，Maldonado らの調査に沿った，SATD の除去までの期間や誰がその除去を行ったかの調査に限らず，それらとは異なる調査から SATD の研究の幅を広げようと考えている．

謝辞

本研究は，JP18H03222 による助成を受けた．

参考文献

[1] Bavota, G. and Russo, B.: A large-scale empirical study on self-admitted technical debt, *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016*, pp. 315–326 (2016).

[2] Curtis, B., Sappidi, J. and Szynkarski, A.: Estimating the size, cost, and types of technical debt, *Proceedings of the Third International Workshop on Managing Technical Debt, MTD 2012*, pp. 49–53 (2012).

[3] da S. Maldonado, E., Abdalkareem, R., Shihab, E. and Serebrenik, A.: An Empirical Study on the Removal of Self-Admitted Technical Debt, *Proceedings of IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, pp. 238–248 (2017).

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

[5] Fontana, F. A., Ferme, V. and Spinelli, S.: Investigating the impact of code smells debt on quality code evaluation, *Proceedings of the Third International Workshop on Managing Technical Debt, MTD 2012*, pp. 15–22 (2012).

[6] Kamei, Y., da S. Maldonado, E., Shihab, E. and Ubayashi, N.: Using Analytics to Quantify Interest of Self-Admitted Technical Debt, *Joint Proceedings of the 4th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2016) and 1st International Workshop on Technical Debt Analytics (TDA 2016) co-located with the 23rd Asia-Pacific Software Engineering Conference (APSEC 2016)*, pp. 68–71 (2016).

[7] Lim, E., Taksande, N. and Seaman, C. B.: A Balancing Act: What Software Practitioners Have to Say about Technical Debt, *IEEE Software*, Vol. 29, No. 6, pp. 22–27 (2012).

[8] 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 Proceeedings, ICSE 2018*, pp. 9–12 (2018).

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

[10] Vassallo, C., Zampetti, F., Romano, D., Beller, M., Panichella, A., Penta, M. D. and Zaidman, A.: Continuous Delivery Practices in a Large Financial Organization, *Proceedings of IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, pp. 519–528 (2016).

[11] Wehaibi, S., Shihab, E. and Guerrouj, L.: Examining the Impact of Self-Admitted Technical Debt on Software Quality, *Proceedings of IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016*, pp. 179–188 (2016).

[12] Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A. E. and Ubayashi, N.: Revisiting the applicability of the pareto principle to core development teams in open source software projects, *Proceedings of the 14th International Workshop on Principles of Software Evolution, IWPSE 2015*, pp. 46–55 (2015).