Vol.2018-SE-200 No.9
2018/12/3

“Waiting” Self-Admitted Technical Debtの分析と考察

田内 遥夏[1,a)]　中才 恵太朗[1,b)]　畑 秀明[1,c)]　松本 健一[1,d)]

概要：ソフトウェア開発において，短期間で開発課題を解決するために，止むを得ず最善ではない方法が選択される場合があり，この解決方法によってもたらせるものは Technical Debt と呼ばれている．Technical Debt が蓄積するとソースコードの複雑性が増し，将来的に開発の遅延やバグの原因になる恐れがある．この悪影響を避けるために Technical Debt の早期返済を促す必要がある．その支援として，自動抽出した返済可能な Technical Debt や原因による分類を活用した返済の優先順位をソフトウェア開発者に提示する方法が考えられる．Technical Debt は意図せずしてソースコードに含まれてしまうこともあるが，ソースコード中でソフトウェア開発者によるコメントを伴って累積するような故意の Technical Debt を，特に Self-Admitted Technical Debt（SATD）と呼ぶ．本研究では，SATD の中でも “Waiting” SATD という，「他の要因」が解決された時，返済可能になる SATD に着目した．Waiting SATD が返済可能になった時，ソフトウェア開発者にそれを通知することは SATD の早期返済の助けになる．一方で，すべての Waiting SATD において「他の要因」は既存の分類体系が存在せず，関数名など具体的な箇所が示されている場合もあるが，多くは不明瞭で，実態が明らかではない．「他の要因」が解決し，返済可能になったかどうかを把握できるようになるためには，「他の要因」の具体的な箇所や原因を明らかにする必要がある．そこで，収集した SATD から人手で Waiting SATD を抽出，精読し，Waiting SATD が返済可能になったことを最終的にソフトウェア開発者に自動で通知できるかを検討するため，「他の要因」となる対象の具体的な箇所と種類を調査した．

1. はじめに

一般的なソフトウェア開発では，納品や発売があり，そのための締切がある．万全なプロジェクトマネジメントが行われている場合，設計通りの開発だけではなく，予定外のバグや外部からの要求の変化など，当初にはなかった開発課題にも余裕をもって対応でき，ソフトウェアはスケジュール通りに納品や発売がされるだろう．しかし，現実的には人材や時間のリソースを限界まで使役するようにスケジュールが組まれ，ソフトウェアの品質と開発速度が天秤にかけられる．締切まで時間が足りない状況では，開発課題を解決するために止むを得ず，最善ではない方法が選択される場合があり，Technical Debt と呼ばれている [2]．本研究は，ソースコード中の Technical Debt を扱う．Technical Debt には，例えば，使用していない関数，バグ回避のためのコード断片などがある．Technical Debt は 10 年以上放置されることもあるが [7]，蓄積するとソースコードの複雑性が増して保守コストが重くなり，将来的に開発の遅延やバグの原因になる恐れがある [5]．このような悪影響を避けるために，ソフトウェア開発者へ Technical Debt の早期返済を促す必要がある．例として，Technical Debt の中でも返済可能なものや原因による分類を活用した返済の優先順位をソフトウェア開発者に提示する方法が考えられる．Technical Debt は，ソフトウェア開発者が意図していないにも関わらず偶然ソースコードに発生してしまうことも，故意にソースコードに含めることもある [4]．Technical Debt の中でも，ソフトウェア開発者によるコメントを伴って故意にソースコードに含まれる，ソフトウェア開発者に自覚がある Technical Debt を，特に Self-Admitted Technical Debt（SATD）と呼ぶ [9]．

本研究では，SATD の中でも “Waiting” SATD という，「他の要因」が解決された時，返済可能になる SATD に着目した．「他の要因」とは，Waiting SATD それ自体とは異なるバグや SATD のことである．分かりやすく例えると，**図 1** のように，開発課題の解決という目標地点に向かう最短経路があるが，最短経路を通れないように塞いでいる岩が「他の要因」であり，岩を回避するための面倒で長い迂回路が Waiting SATD である．Waiting SATD が返済

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[a)] tauchi.haruka.sx7@is.naist.jp
[b)] nakasai.keitaro.nc8@is.naist.jp
[c)] hata@is.naist.jp
[d)] matumoto@is.naist.jp

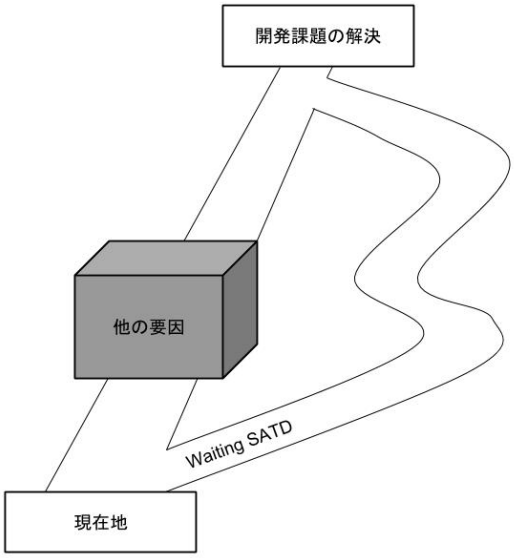

図 1　Waiting SATD と「他の要因」の概念図
Fig. 1　Conceptual diagram of Waiting SATD and “other factors”

可能になるとは，岩が除去されて最短経路が通れるようになること，Waiting SATD を返済するとは，不要になった迂回路を埋めて最短経路だけを残すことである．一方で，Waiting ではない SATD は，最短経路が見つからなかった場合や，最短経路を敷けなかった場合の迂回路である．

これまで説明してきた，Technical Debt, SATD, Waiting SATD の関係は**図 2** のようにまとめられる．Technical Debt のうち，ソフトウェア開発者が自覚を持っているものが SATD であり，SATD のうち，「他の要因」のせいで返済できないものが Waiting SATD である．「他の要因」を解決し，Waiting SATD 自体も返済することで，Waiting SATD の返済は完了する．また，Waiting SATD は「他の要因」が解決された時に返済可能になるので，「他の要因」さえ解決すれば Waiting ではない SATD と同じように扱えるようになるのである．つまり，Waiting SATD を返済するという課題は，ソフトウェア開発者に対して「他の要因」の解決を支援し，円滑に「他の要因」が解決されることで，SATD を返済するという課題と同一視できるようになる．同時に，「他の要因」が解決されたタイミングを検出できれば，ソフトウェア開発者に Waiting SATD が返済可能になったことを通知する機会が得られ，SATD の早期返済の助けになる．一方で，すべての Waiting SATD において，「他の要因」は既存の分類体系が存在していない上に，クラスや関数など具体的な箇所が名指しで示されている場合もあるが，多くは不明瞭で，実態が明らかではない．「他の要因」が解決した後，ソフトウェア開発者が Waiting SATD が返済可能になったことを把握できるようになるためには，「他の要因」の具体的な箇所や種類を明らかにする必要がある．

本研究では，Waiting SATD が返済可能になったことを最終的にソフトウェア開発者に自動で通知できるかを検討するため，「他の要因」の具体的な箇所と種類を分析する．

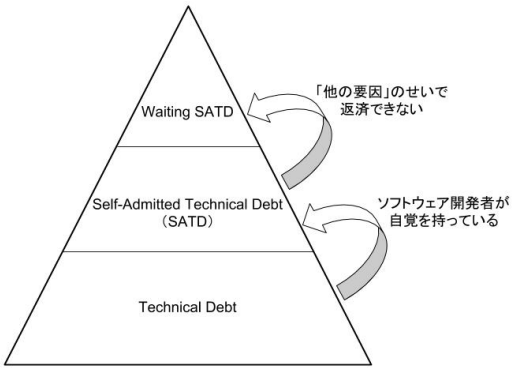

図 2　Techinical Debt, Self-Admitted Technical Debt（SATD）, Waiting SATD の関係
Fig. 2　Relationship of Technical Debt, Self-Admitted Technical Debt (SATD), Waiting SATD

2. “Waiting” Self-Admitted Technical Debt

我々は，Technical Debt のうち「他の要因」のせいでソフトウェア開発者が意図的にソースコードに含めることになったものを Waiting SATD と呼ぶ．意図的であるかどうかは，コメントを伴っているか否かによって判断する．

我々が Waiting SATD の存在に気付いたのは，ソースコードから SATD を示唆するコメントを自動抽出できないか，と Maldonado et al. [6] が公開しているデータセット[*1]（以下，データセットと呼ぶ）を自然言語処理の観点で分析していた時である．「（クラスなど）を修正すれば，ここ（コメント箇所）を削除できる」，「（フレームワーク，プラグイン）がバージョンアップすれば不要」など，コメント以外の箇所について言及してあり，そのコメント以外の箇所が適切に変更された後になってようやくコメント箇所を変更できる旨のコメントが存在していた．コメント以外の箇所，つまり本論文の冒頭から「他の要因」と呼んでいるものが原因で修正できない SATD が存在し，また「他の要因」も修正しなければならないという段階を踏む必要があるために，このような SATD はソフトウェア開発者の記憶や TODO から忘れ去られがちなのではないかと考えた．そこで，「他の要因」が解決したタイミングでソフトウェア開発者に，「他の要因」が解決したので即座に修正できることを通知できれば SATD の早期返済に繋がると期待する．「他の要因」の解決を待っていることから，このような SATD を我々は **“Waiting” Self-Admitted Technical Debt** と呼ぶ．

Technical Debt には設計，テスト，欠陥など，様々な種

[*1] maldonado/tse.satd.data - GitHub.com
https://github.com/maldonado/tse.satd.data

類があり，そのうちの設計の負債の影響が最も大きいことが知られている [1], [8]．データセットの SATD は，設計と要求，文書，テスト，欠陥，その他に分類されており，設計と要求に分類されたデータが多い．また，設計と要求の SATD が一般的とされている [6]．本研究では設計と要求の SATD から Waiting SATD を抽出した．

3. 分析手法

本分析の目的は，Waiting SATD が返済可能になったことを最終的にソフトウェア開発者に自動で通知できるかを検討するために，「他の要因」の具体的な箇所と種類を明らかにすることである．また，分析手順は大きく以下の通りである．

(1) SATD データのフィルタリング
(2) フィルタリングした SATD データから Waiting SATD を抽出
(3) 抽出した Waiting SATD を分析

3.1 分析目的

「Waiting SATD が返済可能になったことを最終的にソフトウェア開発者に自動で通知できるか」を検討するために明らかにすべきことは，**「Waiting SATD が返済可能になったことを検出できるかどうか」**である．前述の**「Waiting SATD が返済可能になったこと」**をより具体的に表現すると，**「Waiting SATD を引き起こす『他の要因』の具体的な箇所とその箇所が適切に修正されたこと」**になる．また，ソフトウェア開発者に自動で通知する方法はソフトウェアの開発プラットフォームに依存し，分析目的に沿わないため論じない．以上より検討内容は，「Waiting SATD が返済可能になったことを最終的にソフトウェア開発者に自動で通知できるか」から「Waiting SATD を引き起こす『他の要因』の具体的な箇所とその箇所が適切に修正されたことを検出できるかどうか」に言い換えられる．さらに，**適切に修正**という条件は単純な**変更**に緩和できるので，**「Waiting SATD を引き起こす『他の要因』の具体的な箇所とその箇所が変更されたことを検出できるかどうか」**となる．**「Waiting SATD を引き起こす『他の要因』の具体的な箇所とその箇所が変更されたことを検出できるかどうか」**では，Waiting SATD が本当に返済可能になったかは不明だが，返済可能になった可能性がある．最終的にソフトウェア開発者に通知すると役に立つ，返済可能になったタイミングは，すべての「他の要因」が変更されたタイミングのうちのいずれかになることは明らかである．したがって，少なくとも**「Waiting SATD を引き起こす『他の要因』の具体的な箇所とその箇所が変更されたことを検出できるかどうか」**が分かるか検討することを分析目的とする．

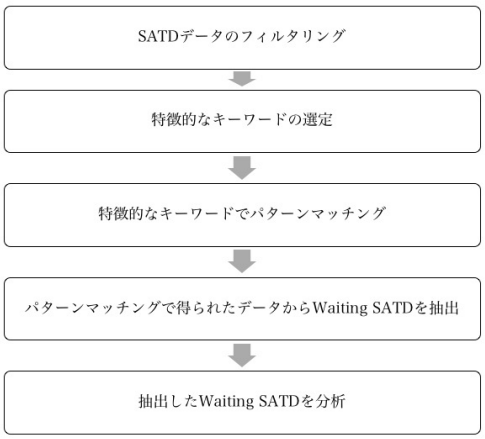

図 3 分析手順
Fig. 3 Analysis procedure

3.2 分析手順

分析手順を**図 3** に示す．まず，データセットの SATD コメントデータをフィルタリングする．次に，フィルタリングした SATD コメントデータから，明確に Waiting SATD に当てはまるコメントを解析し，Waiting SATD に特徴的なキーワードを選定する．さらに，そのキーワードを用いて SATD のコメントデータ全体をパターンマッチングすることで，分析対象とするデータを絞り込む．最後に，パターンマッチングで得られたデータから人手で Waiting SATD を抽出し，分析する．各手順について，以下で詳細に説明する．

本研究では，Maldonado et al. が公開しているデータセットを用いる．このデータセットは，Ant, ArgoUML, Columba, EMF, Hibernate, JEdit, JFreeChart, JMeter, JRuby, SQuirrel SQL と，異なるアプリケーションドメインから 10 のオープンソースプロジェクトを選び，これらのソースコードから SATD のコメントだけを残したものである．データセットには，プロジェクト名，SATD の種類，コメントが含まれており，全 83,144 件である．設計と要求の SATD が一般的とされているため，データセットの設計と要求の SATD を分析対象とする．全データのうち，設計のデータは 2,703 件，要求のデータは 757 件ある．すなわち分析対象のデータは，設計と要求のデータを合わせて計 3,460 件になる．

本研究は Waiting SATD のみを分析することが目的であるため，効率よく Waiting SATD を抽出したい．そこで，分析対象のデータに対してパターンマッチングし，さらにデータを絞り込む．パターンマッチングに使用するキーワードは，分析対象のデータの中から明確に Waiting SATD に当てはまるデータのコメントを解析し，Waiting SATD に特徴的なものを選ぶ．選んだキーワードの一覧が**図 4** である．例えば，キーワード **“remove”** は**図 5**,

when once remove workaround fixed after will

図 4　特徴的なキーワード
Fig. 4　Characteristic keywords

```
/*
Wrap nulls.
This is a bit of a cludge to address a deficiency in the class
generator whereby it does not wrap nulls on method delegate. See
Class Generator.java. If we fix that then we can remove this.
(just have to generate the code there.)
*/
```

図 5　“remove” を含むコメント 1
Fig. 5　Comment including “remove” 1

```
// TODO: Once we have fixed all subclasses the title will
// always be localized so this localization can be removed.
```

図 6　“remove” を含むコメント 2
Fig. 6　Comment including “remove” 2

図 6[*2]などを分析して選んだ．

さらに，パターンマッチングで絞り込まれたデータのコメントを人手で精読し，Waiting SATD データを抽出する．

最後に，抽出した Waiting SATD データのコメントを人手で精読し，「他の要因」となる対象の具体的な箇所と種類を分析する．

4. 分析結果と考察

本章では，3 章で示した分析手法で Waiting SATD を分析した結果について説明する．特徴的なキーワードを用いたパターンマッチングの結果から Waiting SATD の存在割合，また，「他の要因」となる対象の具体的な箇所と種類がどのようであったかを述べる．

4.1 Waiting SATD の存在割合

特徴的なキーワードを用いてパターンマッチングした結果を表 1 に示す．Waiting SATD に特徴的なキーワードでパターンマッチングしたが，分析対象のデータは 3,460 件から 681 件に大きく絞り込まれ，約 20%まで減少した．

パターンマッチングの後，人手で Waiting SATD を抽出した結果，設計の Waiting SATD データは 80 件，要求の Waiting SATD データは 5 件と，さらに減少した．

そこで，Waiting SATD がパターンマッチングして得られたデータ中に存在する割合を，以下の数式によって確かめた．Waiting SATD の件数とは，人手で抽出した Waiting SATD の件数，SATD の件数とは，パターンマッチングで抽出した SATD の件数のことである．

$$WaitingSATD\text{の存在割合} = \frac{WaitingSATD\text{の件数}}{SATD\text{の件数}}$$

SATD，Waiting SATD，Waiting ではない SATD，Waiting SATD の存在割合を，設計のものについては表 2，要求のものについては表 3 にまとめた．データには，特徴的なキーワードを複数含むものも存在するため，「全データ」という重複のない件数も示した．表中の **SATD** とは，特徴的なキーワードでパターンマッチングして抽出した SATD の件数であり，数式の分母にあたる．同じく表中の **Waiting SATD** とは，前述の SATD 中から人手で抽出した Waiting SATD の件数であり，数式の分子にあたる．設計の Waiting SATD も要求の Waiting SATD も，特徴的なキーワードでパターンマッチングして得られたデータの中でも存在割合が低い．設計の Waiting SATD は約 10%〜約 40%，要求の Waiting SATD は 0%〜約 3%しか存在していない．すなわち，パターンマッチングしていないデータでは著しく存在割合が低いと分かる．

4.2 「他の要因」となる対象の分析

4.1 節より，Waiting SATD の件数が SATD の種類を問わず少なかったため，SATD の種類を区別せずに「他の要因」となる対象の具体的な箇所と種類を分析し，表 4 にまとめた．対象の種類としては，**外部サイト**が最も多かった．外部サイトとは，GitHub issue や ORACLE Java Bug Database などのことであり，コメントより詳細である場合が多いため，外部サイトに詳細があるものは外部サイトとして分類した．また，データセットが Java 言語のプロジェクトを対象にしてある影響か，**ソフトウェア開発キット**の JDK が対象の「他の要因」も多く見られた．一方で，**曖昧**な「他の要因」も存在しており，例として図 7 を挙げる．このコメントでは，クラスが「他の要因」の対象になってはいるが，どのクラスなのかが分からず曖昧である．コンピューターを使えば，ソフトウェア開発者はプロジェクトに含まれるすべてのクラスを監視できる．しかし，すべてのクラスの変更の中で「他の要因」の対象である唯一のクラスが変更されている可能性は非常に低い．このように監視の対象が広くなるものを具体的とは言えない．「他の要因」の対象が曖昧ではなく，**不明**なものとして，図 8 などがある．“ This” が指している箇所は分からない上に，この Waiting SATD が返済可能になるのは “fix the problem properly” と曖昧である．どのように修正すれば正しいのかは不明である．しかし，「他の要因」の対象が具体的であるとする，先述のソフトウェア開発キットやフレームワーク，プラグインの場合に関しても，バージョンが指定されているものと指定されていないものがあった．いずれもバージョン指定がなくとも一意に定まると考えられるが，バージョン指定されているほうが些末な変更を拾わずに済

[*2] 見やすいように適宜，改行を挿入している

表 1　特徴的なキーワードを用いたパターンマッチングでヒットした SATD の件数
Table 1　Number of SATDs hit by pattern matching using distinctive keywords

	合計	when	once	remove	workaround	fixed	after	will
設計	611	116	42	129	61	14	35	115
要求	70	25	8	18	1	4	6	21

表 2　設計 Waiting SATD の存在割合
Table 2　Presence ratio of Waiting design SATD

	全データ	when	once	remove	workaround	fixed	after	will
SATD	611	116	42	129	61	14	35	115
Waiting SATD	80	18	12	20	24	3	13	15
Non Waiting	531	98	30	109	37	11	22	100
存在割合	0.131	0.155	0.286	0.155	0.393	0.214	0.371	0.130

```
// Remember to change this when the class changes ...
```

図 7「他の要因」の対象が曖昧なコメント
Fig. 7　Comments with ambiguous subjects of “other factors”

```
// TODO: This seems like a brute force workaround (and a very
// indirect one at that). It appears to be needed though until
// we fix the problem properly. - tfm 20070904
```

図 8「他の要因」の対象が不明なコメント
Fig. 8　Comment of unknown target of “other factors”

むため，Waiting SATD が返済可能になった変更のタイミングを検出するコストが低くなる．

4.3　「Waiting SATD を引き起こす『他の要因』の具体的な箇所とその箇所が変更されたことを検出できるかどうか」が分かるかの検討

以上の分析結果と考察より，「Waiting SATD を引き起こす『他の要因』が変更されたことを検出できるかどうか」は，「他の要因」となる対象の具体的な箇所や種類が把握できれば，分かると考えられる．なぜなら，具体的な箇所が分かれば，そこを監視することで「他の要因」の変更を検出できるからである．一方で，ソフトウェア開発者への通知が現実的に実現可能かどうかを考慮しなければ，図 7 のように「他の要因」となる対象が曖昧で，箇所は分からずとも種類が分かる場合にも検出できる可能性が高いことが分かった．ただし，図 8 のように対象が不明の場合は検出が不可能だと考えられる．

4.4　妥当性への脅威

本論文の分析手法の妥当性の論点として，以下の項目が挙げられる．それぞれに対して続く項で議論する．

- コメントに基づいて Waiting SATD と判断していいのか
- 古いコメントが残っている可能性はないのか

4.4.1　コメントに基づいて Waiting SATD と判断していいのか

Technical Debt に関しては，コメントよりも定量的な指標にしたがってソースコードから判断したほうがよいと思える．しかし，SATD はソフトウェア開発者の自覚が必要であり，自覚があるかどうかを見極めるためにソフトウェア開発者自身が書いたコメントを利用することは，コメントを通じて Technical Debt を識別でき，SATD に関係していることが知られており [9]，妥当だと考える．Waiting SATD も SATD の一種であるため，同様に妥当な上に，「他の要因」の解決を待っているかどうかはソースコードだけでは分からないので，むしろコメントが必須だと考える．ソフトウェア開発者によって書かれたコメントは，ソースコード中で最もソースコード理解の信憑性が高く，情報量が多いものだと考えられ，Waiting SATD の解析に役に立つ．

また，ソースコードのコメントはソースコードファイルから正規表現を使って簡単かつ効率的に抽出できる [6] ため，今後のデータ拡充を考慮するとソースコードのコメントを利用していくことが妥当である．

4.4.2　古いコメントが残っている可能性はないのか

ここでいう古いコメントとは，単にそのコメントが残されたのが昔という意味ではなく，コメントに対応するソースコードが残っていないようなコメントを指す．古いコメントが残っている可能性がないとは言い切れないが，不要になった Waiting SATD を処理する際には，それに付随しているコメントを目安にすると考えられるので，コメントが残る可能性は低い．また，ソースコードの変更がコメントの変更と一致していることも示されている [3], [9]．他にも，**図 9** のように Waiting SATD コメント自体の削除を促しているものも存在していた．

5.　まとめと今後の課題

本研究の分析を通じて，Waiting SATD は非常に少ないが存在していると分かった．しかし，その少なさは著しく，

表 3　要求 Waiting SATD の存在割合
Table 3　Presence ratio of Waiting implementation SATD

	全データ	when	once	remove	workaround	fixed	after	will
SATD	70	25	8	18	1	4	6	21
Waiting SATD	5	2	2	3	0	0	0	1
Non Waiting	65	23	6	15	1	4	6	20
存在割合	0.071	0.08	0.25	0.167	0	0	0	0.048

表 4　「他の要因」の対象の種類と件数
Table 4　Type and number of objects of “other factors”

対象の種類	件数
外部サイト	15
ソフトウェア開発キット	8
フレームワーク，プラグイン	8
コンストラクタ	8
クラス	5
関数	4
OS	3
ソースコード文	3
サブクラス	2
ライブラリ	2
IDE	1
DBMS	1
ソースコードファイル全体	1
TODO	1
曖昧	10
不明	17

```
//TODO: remove when code below in characters() is removed
//private static final String RETURNSTRING = “\n”;
```

図 9　削除を促すコメント
Fig. 9　Comment to Remove

データセットのうち，分析した設計と要求の SATD データが 611 件であることに対し，85 件しか存在していなかった．しかし，Waiting SATD を引き起こす「他の要因」となっている対象について, ほとんどの Waiting SATD データのコメントには具体的にコメントされていた．したがって，Waiting SATD データのコメントをソースコードから抽出できた場合，多くの Waiting SATD において，「他の要因」の変更を監視することは実現可能性が高いと言える．Waiting SATD の「他の要因」の変更や解決によって Waiting SATD が返済可能になったことを自動的にソフトウェア開発者に通知できれば，Waiting SATD の早期返済に役立つ．したがって，Waiting SATD は多く抽出できるようになればなるほど価値があると言える．今後の課題として，機械学習などを用いて Waiting SATD を抽出できるようにしていきたい．

謝辞　本研究は JSPS 科研費 16H05857 と 17H00731 の助成を受けた．

参考文献

[1] Alves, N. S. R., Ribeiro, L. F., Caires, V., Mendes, T. S. and Spnola, R. O.: Towards an Ontology of Terms on Technical Debt, *2014 Sixth International Workshop on Managing Technical Debt*, pp. 1–7 (online), DOI: 10.1109/MTD.2014.9 (2014).
[2] Cunningham, W.: The WyCash Portfolio Management System, *SIGPLAN OOPS Mess.*, Vol. 4, No. 2, pp. 29–30 (online), DOI: 10.1145/157710.157715 (1992).
[3] Fluri, B., Wursch, M. and Gall, H. C.: Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes, *14th Working Conference on Reverse Engineering (WCRE 2007)*, pp. 70–79 (online), DOI: 10.1109/WCRE.2007.21 (2007).
[4] Fowler, M.: TechnicalDebtQuadrant, (online), available from ⟨https://martinfowler.com/bliki/TechnicalDebtQuadrant.html⟩ (accessed 2018-11-03).
[5] 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 (online), DOI: 10.1109/MS.2012.130 (2012).
[6] 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 (online), DOI: 10.1109/TSE.2017.2654244 (2017).
[7] Maldonado, E., Abdalkareem, R., Shihab, E. and Serebrenik, A.: An Empirical Study On the Removal of Self-Admitted Technical Debt, Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME’17), IEEE (2017).
[8] Marinescu, R.: Assessing technical debt by identifying design flaws in software systems, *IBM Journal of Research and Development*, Vol. 56, No. 5, pp. 9:1–9:13 (online), DOI: 10.1147/JRD.2012.2204512 (2012).
[9] Potdar, A. and Shihab, E.: An Exploratory Study on Self-Admitted Technical Debt, *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 91–100 (online), DOI: 10.1109/ICSME.2014.31 (2014).