修士論文

外部事象に起因する技術的負債についての ソースコードコメント分析

田内 遥夏

2019年3月15日

奈良先端科学技術大学院大学
情報科学研究科

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

田内 遥夏

審査委員：

松本 健一 教授　（主指導教員）
飯田 元 教授　（副指導教員）
石尾 隆 准教授　（副指導教員）
畑 秀明 助教　（副指導教員）

外部事象に起因する技術的負債についての ソースコードコメント分析*

田内 遥夏

内容梗概

ソフトウェア開発において，短期間で開発上の課題を解決しなければならない場合，ソフトウェア開発者はコメントを付して最善ではない解決方法，すなわちワークアラウンドを使用することがある．ワークアラウンドが使用される理由は，開発速度をあまり低下させずに課題解決できるからである．しかし，ワークアラウンドは，長期的には生産性の低下，開発課題の増加などの悪影響を引き起こす解決方法であり，技術的負債の一つとして知られている．技術的負債による悪影響を未然に防ぐためには，技術的負債の計画的な早期返済を促す必要がある．

本研究では，外部事象に存在する「他の要因」の解消を待機しているために返済できない技術的負債（待機型技術的負債）に着目する．現在，すべての待機型技術的負債における,「他の要因」の分類体系は存在していないため，本研究では待機型技術的負債に関するソースコードコメントに基づいた分類と分析を行う．その結果,「他の要因」の分類によって待機型技術的負債を返済する優先順位を決められる可能性が高いことを明らかにした．この分類基準が待機型技術的負債を計画的に早期返済する支援に役立つと考えられる．

キーワード

技術的負債, ソースコードコメント, ソフトウェア品質, 実証研究, オープンソースソフトウェア

*奈良先端科学技術大学院大学 情報科学研究科 修士論文, 2019 年 3 月 15 日.

A Study of Self-Admitted Technical Debt Attributable to External Events*

Haruka Tauchi

Abstract

In software development, if development challenges must be solved in a short period of time, software developers may use comments and non-optimal solution, i.e. workaround. The reason why the workaround is used is that the problem can be solved without reducing development speed too much. However, workaround causes adverse effects such as a decrease in productivity and an increase in development tasks in the long term, and it is known as one of Self-Admitted Technical Debts (SATD). In order to prevent adverse effects of SATD beforehand, it is necessary to prompt the planned early repayment of SATD.

In this research, We focus on SATD that can not be repaid because it is waiting for elimination of "other factors" existing in external events (On-Hold SATD). Currently, since there is no classification system of "other factors" in all on-hold SATD, this research classifies and analyzes based on source code comments on on-hold SATD. As a result, it is clear that there is a high possibility of deciding the priority order to repay on-hold SATD by classification of "other factors". It is thought that this classification standard will be useful for assisting early repayment of on-hold SATD systematically.

Keywords:

*Master's Thesis, Graduate School of Information Science, Nara Institute of Science and Technology, March 15, 2019.

Self-admitted technical debt, Source code comment, Software quality, Empirical Study, Open source software

目 次

図 目 次

表 目 次

1. はじめに

1.1 研究背景および関連研究

一般的なソフトウェア開発では，消費者や依頼主など顧客の要求からソフトウェアの目的と仕様を決定したのち，ソフトウェア設計が行われる．開発中には，設計通りの開発だけではなく，開発途中に発生する予定外のバグ，顧客からの要求変化など，当初の設計には存在していない開発課題が発生することがある．開発課題の解決には時間が必要だが，ソフトウェア開発には発売や納品に向けた締切があるため，開発スケジュールを大幅に変更しない程度の時間の範囲で課題解決を試みることになる．また，開発課題は小規模なものから大規模なものまで複数発生する可能性がある．すべての課題解決に使う時間の合計を開発スケジュールを大幅に変更しない程度の時間に抑える必要があるため，それぞれの開発課題に使える時間は多くない．前述のように課題解決に使える時間が限られているソフトウェア開発現場では，課題解決も含めた開発速度とソフトウェア品質とがトレードオフの関係になる [1]．開発速度とソフトウェア品質がトレードオフの関係になった結果，ソフトウェア開発者は開発速度のためにワークアラウンドを使用して課題解決することがある [1]．ワークアラウンドを使用する課題解決は開発速度をあまり低下させないが，将来的に悪影響を与える技術的負債を残してしまう [2]．

技術的負債とは，Cunningham [3] が導入した比喩である．技術的負債は，金融における負債と同様に，将来の開発速度を前借りしてその場しのぎで開発課題を解決できる．しかし，利子として技術的負債による将来的な悪影響を受けることになる．

開発速度を借りるために抱えた技術的負債が引き起こす将来的な悪影響は 2 種類ある [4]．一つ目は，技術的負債を抱えている期間中，ソフトウェア開発において生産性が低下することである．生産性が低下する具体的な例として，単体テストを作成しないという技術的負債が挙げられる．単体テストが存在しない場合，新しい開発，テスト，デバッグに多くの時間がかかるようになり，生産性が低下する．二つ目は，技術的負債を返済する際には，ワークアラウンドを使用せず開

発課題を実装していた場合よりも，実装時間が多く必要になることである．例えば，バグを避けるために抱えた技術的負債を返済するために，避けたバグを修正すると異なる箇所にバグが発生する場合がある．技術的負債を抱えずにバグを修正していれば，修正するバグは 1 箇所だけで済み，実装時間が少なかった可能性が高い．また，技術的負債はソフトウェア開発の中で悪循環を引き起こす可能性がある．具体的には，開発状況がスケジュールより遅れることで技術的負債を残し，残された技術的負債が生産性を低下させ，生産性の低下によって新機能の開発期間が短縮されるのでさらに開発状況がスケジュールより遅れ，遅れを挽回するために技術的負債を残す，といった悪循環になる可能性がある．ただし，十分に早く技術的負債を返済した場合は将来的な悪影響が発生しない [4]．しかし，現実的には 10 年以上放置されることもあり [5]，技術的負債を多く抱えた結果，生産性が低下したソフトウェアプロジェクトがある．

技術的負債は将来的な悪影響を引き起こすが，投資の観点では有益な側面がある [6]．ソフトウェアの市場投入が早ければ早いほど，早期にユーザーからフィードバックを受けてソフトウェアを改善できる．このため，技術的負債を抱えるが開発速度をあまり低下させずに課題解決することは市場投入までの時間を大幅に短縮でき，有益である．技術的負債の有益な側面を活用するためには，ソフトウェア開発において，技術的負債を抱える課題解決を，バグ修正や新機能の開発と同様に市場投入のための重要な戦略の一部に位置付ける必要がある．実際に，技術的負債を可視化する，技術的負債を計画に組み込む，技術的負債と将来の悪影響を関連づけて返済する，という取り組みを行なっている組織は，技術的負債を抱えて課題解決しても成功している [2]．

技術的負債は，将来的な悪影響を引き起こすが，早い段階でソフトウェアを市場投入できるという有益な側面もある．技術的負債の有益な側面を活用するために技術的負債を導入し，将来的な悪影響を発生させずに返済できるならば，技術的負債によってソフトウェアの生産性を高めることができる．技術的負債によってソフトウェアの生産性を高めるためには，技術的負債を明確に管理し，計画的に早期返済することが重要である．

しかし，実際には，技術的負債は，ソフトウェア開発者が意図していないにも

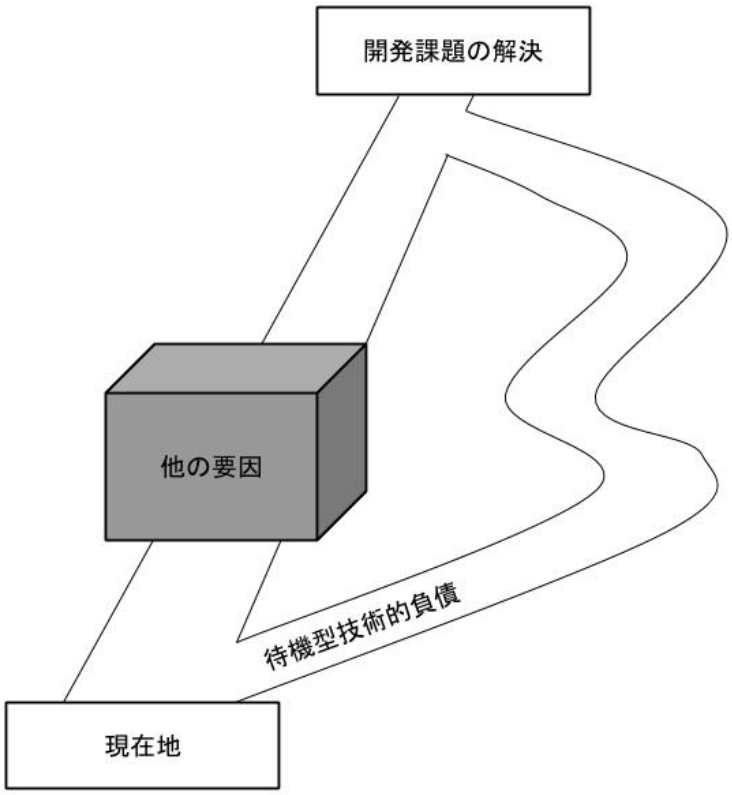

図 1 外部事象に起因した技術的負債の概念図

関わらず偶然ソースコードに発生してしまうことも，意図的にソースコードに含めることもある [7]．技術的負債の中でも，ソフトウェア開発者によるコメントを伴ってソースコードに含まれるものは，ソフトウェア開発者が意図的に導入した技術的負債であることが分かっている [8]．本論文内で単に技術的負債と言う場合は，特別な場合を除き，ソフトウェア開発者が意図的に導入した技術的負債を指す．

本研究では，外部事象に起因した「他の要因」が解決された時，返済可能になる技術的負債 [9] に着目した．外部事象に起因した技術的負債は「他の要因」の修正を待機しているため，本論文内では**待機型技術的負債**と呼ぶ.「他の要因」とは，待機型技術的負債それ自体とは異なるバグや待機型技術的負債のことである．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

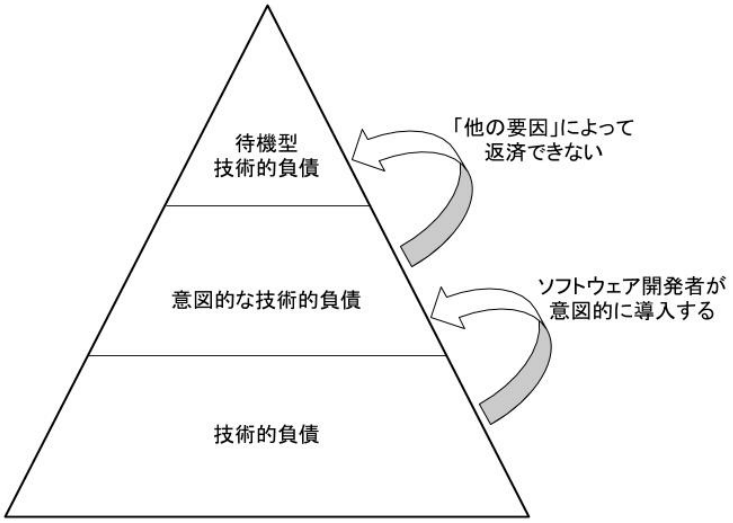

図 2 技術的負債の包含関係

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待機型技術的負債である．待機型技術的負債が返済可能になるとは，岩が除去されて最短経路が通れるようになること，待機型技術的負債を返済するとは，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，外部事象に起因していない技術的負債は，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．

技術的負債, 意図的な技術的負債, 待機型技術的負債の関係は図 2 のようにまとめられる．技術的負債の中には，ソフトウェア開発者が意図的に導入した技術的負債が存在し，意図的な技術的負債の中でも,「他の要因」のせいで返済できないものが待機型技術的負債である.「他の要因」を解決し，待機型技術的負債自体も返済することで，待機型技術的負債の返済は完了する．また，待機型技術的負債は「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」さえ解決すれば外部事象に起因していない技術的負債と同じように扱えるようになるのである．

「（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実際に存在していた．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技術的負債が存在し，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，待機型技術的負債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られる．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待機型技術的負債の早期返済に繋がると期待する．

つまり，待機型技術的負債を返済するという課題は，「他の要因」が解決されることで，待機型ではない技術的負債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待機型技術的負債が返済可能になったことを通知する機会が得られ，待機型技術的負債の早期返済の助けになる．一方で，すべての待機型技術的負債において，「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待機型技術的負債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．

1.2 研究目的

本研究の目的は，待機型技術的負債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」の具体的な箇所と原因を分析することである．待機型技術的負債は「他の要因」の解決を待機している状態の技術的負債であるため，将来的には「他の要因」が解決した状態に遷移する．「他の要因」が解決した状態に遷移した時，待機型技術的負債が返済できようになったことをソフトウェア開発者に通知できれば，「他の要因」によって返済できずに忘れ去られていた待機型技術的負債をソフトウェア開発者に思い出させることができるので，早期返済を促すことができると考えられる．待機型技術的負債は，「他の要因」が解決しない限り返済できないが，返済できるように

なる時点が存在するので，他の技術的負債よりも返済を諦めずに使用できる技術的負債と言える．したがって，ソフトウェア開発者に技術的負債の計画的な早期返済を促すには最適な対象である．

「他の要因」が解決した状態に遷移したことを検知するためには,「他の要因」の具体的な箇所と原因が分かる必要がある．なぜなら,「他の要因」の具体的な箇所が分かることで,「他の要因」が変更され，解決した可能性を検知できるからである．また，原因が分かることで,「他の要因」の解決を待機する以外の方法による返済可能性を提示できるようになると考えられる．しかし，待機型技術的負債は新しく研究され始めたばかりであり,「他の要因」の分類体系が存在していない．そこで，本研究では，負債の計画的な早期返済を促進するために，待機型技術的負債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．

1.3 本論文の構成

本論文は，全 5 節で構成されており，第 2 節では外部事象に起因した技術的負債の特徴を分析するための目的と手法について説明する．分析結果は第 3 節で詳説し，第 4 節で分析結果の考察と妥当性への脅威を議論する．最後に，第 5 節で本論文のまとめと今後の課題を述べる．

2. 分析手法

本分析の目的は，待機型技術的負債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. 技術的負債データのフィルタリング
2. フィルタリングした技術的負債データから待機型技術的負債を抽出
3. 抽出した技術的負債を分析

2.1 分析目的

「待機型技術的負債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,「待機型技術的負債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「待機型技術的負債が返済可能になったこと」をより具体的に表現すると,「待機型技術的負債を引き起こす『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,「待機型技術的負債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「待機型技術的負債を引き起こす『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,「待機型技術的負債を引き起こす『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.「待機型技術的負債を引き起こす『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，待機型技術的負債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「待機型技術的負債を引き

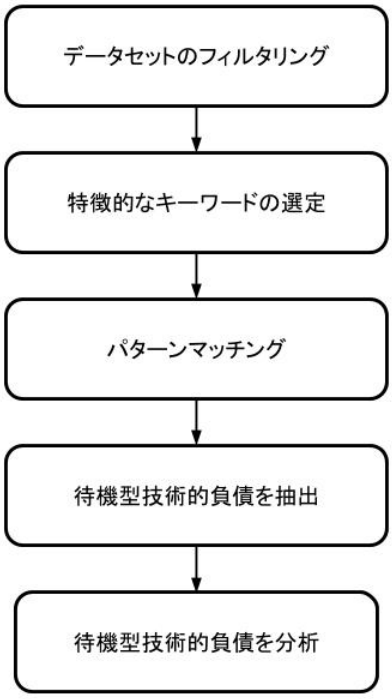

図 3 分析手順

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．

2.2 分析手順

分析手順を図3に示す．まず，データセットの技術的負債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技術的負債コメントデータから，明確に待機型技術的負債に当てはまるコメントを解析し，待機型技術的負債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技術的負債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待機型技術的負債を抽出し，分析する．各手順について，以下で詳細に説明する．

表 1 特徴的なキーワード

when
once
remove
workaround
fixed
after
will

本研究では，Maldonado et al. [1] が公開しているデータセット[1]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技術的負債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，技術的負債の種類，コメントが含まれており，全 83,144 件である．技術的負債には設計，テスト，欠陥など，様々な種類があり，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 [10]．データセットの技術的負債は，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，設計と要求の技術的負債が一般的とされている [11]．データセットの設計と要求の技術的負債を分析対象とする．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．

本研究は待機型技術的負債のみを分析することが目的であるため，効率よく待機型技術的負債を抽出したい．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は，Maipradit ら [9] による待機型技術的負債に含まれる特徴的なキーワードを使用した．パターンマッチングに使用した，待機型技術的負債に含まれる特徴的なキーワードの一覧を表

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

1 に示す.

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，待機型技術的負債データを抽出する.

最後に，抽出した待機型技術的負債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.

表 2 分析対象の技術的負債の件数

	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

3. 分析結果

本節では，2 節で示した分析手法で待機型技術的負債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分析対象とした待機型技術的負債の存在割合，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．

分析対象データ

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 2 に示す．待機型技術的負債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%になった．

パターンマッチングの後，人手で待機型技術的負債を抽出した結果，設計の待機型技術的負債データは 80 件，要求の待機型技術的負債データは 5 件となった．

そこで，待機型技術的負債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．待機型技術的負債の件数とは，人手で抽出した待機型技術的負債の件数，技術的負債の件数とは，パターンマッチングで抽出した技術的負債の件数のことである．

$$\text{待機型技術的負債の存在割合} = \frac{\text{待機型技術的負債の件数}}{\text{技術的負債の件数}}$$

技術的負債，待機型技術的負債，待機型ではない技術的負債それぞれの件数および待機型技術的負債の存在割合を，設計のものについては表 3，要求のものについては表 4 にまとめた．1 件のデータに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**技術的負債**と

表 3 設計の待機型技術的負債の存在割合

	全データ	when	once	remove	workaround	fixed	after	will
技術的負債	611	116	42	129	61	14	35	115
待機型	80	18	12	20	24	3	13	15
非待機型	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

表 4 要求の待機型技術的負債の存在割合

	全データ	when	once	remove	workaround	fixed	after	will
技術的負債	70	25	8	18	1	4	6	21
待機型	5	2	2	3	0	0	0	1
非待機型	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

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技術的負債の件数であり，数式の分母にあたる．同じく表中の**待機型技術的負債**とは，前述の技術的負債中から人手で抽出した待機型技術的負債の件数であり，数式の分子にあたる．設計の待機型技術的負債も要求の待機型技術的負債も，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．設計の待機型技術的負債は約 10%～約 40%，要求の待機型技術的負債は 0%～約 3%しか存在していない．

「他の要因」となる対象箇所

待機型技術的負債の件数が技術的負債の種類を問わず少なかったため，技術的負債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 5 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

表 5 「他の要因」の対象の種類と件数

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
データベース管理システム	1
ソースコードファイル全体	1
TODO	1
曖昧	10
不明	17

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，例として表 6 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，表 7 などがある．“ This” が指している箇所は分からない上に，この待機型技術的負債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

表 6 対象が曖昧なコメント

Remember to change this when the class changes ...

表 7 対象が不明なコメント

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.

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むため，待機型技術的負債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．

「他の要因」となる原因

「他の要因」の原因についても分析結果は表 8 の通りになった．

原因に関しては，3 種類の観点で分類し，分析した．一つ目は，分析対象のコメントが含まれているプロジェクトの外部，内部，どちらか不明のいずれかに原因が存在しているかで分類した（種類 1）．二つ目は，原因の理由が，更新されたバージョンのリリース待ち，バグ修正のどちらかで分類した（種類 2）．三つ目は，バグ修正が原因の場合，修正案があるかどうか，さらに分類した（種類 3）．種類 1〜3 について分類する際には，ソースコードやコメント内の URL などから情報を得ず，コメントから読み取れる情報のみを用いた．なぜなら，将来的に自動的に分類する場合には，コメントの情報のみに基づくと考えられるからである．それぞれの分類に関して，実際のコメントを表 9 に提示する．

表 8 「他の要因」の原因の種類と件数

原因の種類 1	原因の種類 2	原因の種類 3	件数
外部	リリース待ち		17
	バグ	修正案あり	9
		修正案なし	3
内部	リリース待ち		30
	バグ	修正案あり	10
		修正案なし	3
不明	リリース待ち		1
	バグ	修正案あり	10
		修正案なし	2

表 9 「他の要因」の原因の分類におけるコメント

原因の種類 1	原因の種類 2	原因の種類 3
外部	リリース待ち	
The following ugly conversion from text to Byte is necessary because the Byte class is inconsistant. When asked to output as Hex, it does so as an UNSIGNED byte, but when asked to read back the same thing using the Hex radix, it insists that the input must be SIGNED. To get around this, we up-size the conversion to Integer, then truncate that to a byte, and finally convert the byte to a Byte.		
外部	バグ	修正案あり
Don't use tinyint for now, even though Mckoi "supports" it. It's notion of tinyint is 7-bit (not 8-bit) so it is not compatible with other DBs and leads to overflow (resulting in negative values which are a corruption of the actual value inserted/updated). This is not a great work-around. I filed a bug report on the mailing list; hopefully it will get fixed soon.		

外部	バグ	修正案なし
broken Eclipse workaround!		
内部	リリース待ち	
remove this once ComponentMetamodel is complete and merged		
内部	バグ	修正案あり
We currently delete the old values before setting to something new. This is a workaround to issue 6056. We should consider giving an API to get the lower and upper values so that controls can listen directly to those rather than the element containing those values.		
内部	バグ	修正案なし
A FigNodeModelElement with no owner should match here. This is a temporary solution due to FigPool extending FigNodeModelElement when in fact it should not do so.		
不明	リリース待ち	
We're deleting the last diagram so lets create a new one. Once we go MDI we won't need this.		
不明	バグ	修正案あり
hack to ensure the newly created awt fonts will be rendered with the same height as the swt one		
不明	バグ	修正案なし
workaround for hang if match was zero-width. not sure if there is a better way to handle this.		

原因がプロジェクトの外部にあり，リリース待ちをしている場合のコメントは，Java の Byte クラスにおいて挙動が矛盾しているという内容である．プログラミング言語に原因が存在しているため，プロジェクトの外部に分類した．また，Byte クラスの挙動の矛盾が解消されたバージョンがリリースされた時にこの待機型技術的負債は返済できる．

原因がプロジェクトの外部にあり，バグが原因であるが，修正案がある場合のコメントは，データベース管理システム Mckoi がデータ型 tinyint と互換性がないので，バグレポートを提出したという内容である．SQuirrel SQL プロジェクト内のコメントであるため，Mckoi はプロジェクトの外部にあたる．また，バグレポートを提出していることから原因がバグであること，またすぐに修正されると書いてあることから修正案があると推測できる．

原因がプロジェクトの外部にあり，バグが原因であるが，修正案がない場合のコメントは，統合開発環境 Eclipse が壊れた場合のワークアラウンドがあるという内容である．JEdit プロジェクトのコメントであるため，Eclipse はプロジェクトの外部にあたる．また，“broken” という表現から原因がバグであること，また他に言及がないことから修正案がないと言える．

原因がプロジェクトの内部にあり，リリース待ちをしている場合のコメントは，ComponentMetamodel クラスが完成してマージされたら待機型技術的負債に該当する箇所を削除するという内容である．特別にライブラリ名などが指定されていないので，ComponentMetamodel クラスはプロジェクト内部に作成されるクラスだと考えられる．また，ComponentMetamodel クラスが完成してマージされたバージョンがリリースされた時にこの待機型技術的負債は返済できる．

原因がプロジェクトの内部にあり，バグが原因であるが，修正案がある場合のコメントは，6056 番のイシューに報告されているバグの回避策であるという内容である．特別にプロジェクト名が指定されずにイシュー番号が書いてあるので，外部のプロジェクトではなく，内部のイシュー番号であると考えられる．また，イシューに報告されていることからバグである可能性と修正案が提案されている可能性が高い．仮に，修正案が提案されていなくとも，イシューで議論が進むため，時間経過と共に修正案が提案されると推測できる．

原因がプロジェクトの内部にあり，バグが原因であるが，修正案がない場合のコメントは，FigNodeModelElement クラスを拡張することによって FigPool クラスの想定外の挙動を一時的に回避しているという内容である．特別にライブラリ名などが指定されていないので，FigNodeModelElement クラスおよび FigPool クラスはプロジェクト内部に作成されているクラスだと考えられる．また，想定外

の挙動が原因なのでバグと言えること，また他に言及がないことから修正案がないと言える．

原因がプロジェクトの外部，内部のどちらか不明であり，リリース待ちをしている場合のコメントは，MDI (Multiple Document Interface) に移行すると待機型技術的負債に該当する箇所が不要になるという内容である．使用している外部のプロジェクトまたはプロジェクト内部のどちらにおいて MDI に移行することを指しているのか言及がないため，原因は不明とした．MDI に移行することで待機型技術的負債に該当する箇所が不要になり，削除できるようになるので，MDI に移行したバージョンのリリースを待っていると言える．

原因がプロジェクトの外部，内部のどちらか不明であり，バグが原因であるが，修正案がある場合のコメントは，AWT (Abstract Window Toolkit) のフォントが SWT (Standard Widget Toolkit) のフォントと高さが揃わないという内容である．フォントの高さが揃わない原因がプロジェクトの外部である AWT または SWT に存在しているのか，プロジェクトの内部のコードに存在しているのか判断できないため，原因は不明とした．フォントの高さが揃わないという想定外の挙動が原因なのでバグと言えること，またフォントの高さが揃うという想定の挙動があるから修正案があると言える．

原因がプロジェクトの外部，内部のどちらか不明であり，バグが原因であるが，修正案がない場合のコメントは，ハングする条件のワークアラウンドという内容である．条件を満たした場合に，ハングする原因がプロジェクトの外部または内部のどちらにあるか言及がないので，原因は不明とした．想定外のハングが発生するのでバグと言えること，またより良い処理方法があるかどうか分からないと書いてあることから修正案がないと言える．

4. 考察と妥当性への脅威

本節では，3 節で分析した,「他の要因」となる対象箇所と原因に関してそれぞれ考察する．また，本分析における妥当性への脅威を議論する．

4.1 考察

「他の要因」となる対象箇所

「待機型技術的負債を引き起こす『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，表 6 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，表 7 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．

「他の要因」となる原因

「他の要因」となる原因の分析結果から,「他の要因」の対象箇所の分析と合わせて，待機型技術的負債に対してソフトウェア開発者がどのようなアプローチをできるか考えられる.「他の要因」の対象箇所が具体的かどうかを問わず,「他の要因」がプロジェクト外部に存在している場合,「他の要因」を直接解決することはできないため，待機し続けることになるが,「他の要因」がプロジェクト内部に存在している場合,「他の要因」を直接解決することも可能である．原因がリリース待ちまたはバグという点における分析では,「他の要因」が解決するまでの期間を大きく把握できると考えられる．また，バグ修正を待機している待機型技術的負債では，修正案があるから待っているだけでいいのか，修正案がないので修正案の提案や修正まで多くの時間が必要なのか分かる．ある待機型技術的負債に注目した時，待つしかないのか，自分から返済に向けて動けるのかが判断できる．

4.2 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待機型技術的負債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

コメントに基づいて待機型技術的負債と判断していいのか

意図的ではない技術的負債に関しては，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，意図的な技術的負債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，コメントを通じて技術的負債を識別でき，意図的な技術的負債に関係していることが知られている [8] ため，妥当だと考える．待機型技術的負債も意図的な技術的負債の一種であるため，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，待機型技術的負債の解析に役に立つ．

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 [1] 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．

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，不要になった待機型技術的負債を処理する際には，それに付随しているコメントを目安にする

表 10 削除を促すコメント

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

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 [8, 12]．他にも，表 10 のように待機型技術的負債の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. おわりに

本研究の分析を通じて，待機型技術的負債は非常に少ないが存在していると分かった．その少なさは著しく，データセットのうち，分析した設計と要求のデータが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，待機型技術的負債を引き起こす「他の要因」となっている対象について, ほとんどの待機型技術的負債に関するコメントは具体的であると明らかになった．この結果より，待機型技術的負債のコメントをソースコードから抽出できた場合，多くの待機型技術的負債において,「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．待機型技術的負債の「他の要因」の変更や解決によって待機型技術的負債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，技術的負債の計画的な早期返済に役立つ．したがって，待機型技術的負債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，設計と要求の待機型技術的負債で，待機型技術的負債に特徴的なキーワードが異なる可能性があるので，特徴的なキーワードのさらなる調査が必要である．また,「他の要因」が解決した後にどのようにソースコードを編集して待機型技術的負債を返済するのかという観点での分析を行いたい．機械学習などにより待機型技術的負債の待機要因の箇所と原因を自動抽出し，待機要因の箇所と原因で自動分類をすることも期待される．

参考文献

[1] Everton da S. Maldonado, Emad Shihab, and Nikolaos Tsantalis. Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Transactions on Software Engineering*, Vol. 43, No. 11, pp. 1044–1062, November 2017.

[2] Philippe Kruchten, Robert L. Nord, Ipek Ozkaya, and Davide Falessi. Technical Debt: Towards a Crisper Definition Report on the 4th International Workshop on Managing Technical Debt. *SIGSOFT Software Engineering Notes*, Vol. 38, No. 5, pp. 51–54, August 2013.

[3] Ward Cunningham. The WyCash Portfolio Management System. *SIGPLAN OOPS Messenger*, Vol. 4, No. 2, pp. 29–30, December 1992.

[4] Steve McConnell. Managing Technical Debt (slides) in Workshop on Managing Technical Debt (part of ICSE 2013), May 2013.

[5] Everton da S. Maldonado, Rabe Abdalkareem, Emad Shihab, and Alexander Serebrenik. An Empirical Study On the Removal of Self-Admitted Technical Debt. In *Proceedings of the 33rd International Conference on Software Maintenance and Evolution*, ICSME 2017, September 2017.

[6] Yuepu Guo and Carolyn Seaman. A Portfolio Approach to Technical Debt Management. In *Proceedings of the Second Workshop on Managing Technical Debt*, MTD 2011, pp. 31–34, May 2011.

[7] Martin Fowler. Technical Debt Quadrant. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html.

[8] Aniket Potdar and Emad Shihab. An Exploratory Study on Self-Admitted Technical Debt. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution*, ICSME 2014, pp. 91–100, September 2014.

[9] Rungroj Maipradit, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. Wait For It: Identifying "On-Hold" Self-Admitted Technical Debt. *arXiv e-prints*, p. arXiv:1901.09511, January 2019.

[10] Nicolli S. R. Alves, Leilane F. Ribeiro, Vivyane Caires, Thiago S. Mendes, and Rodrigo O. Spínola. Towards an Ontology of Terms on Technical Debt. In *Proceedings of the 2014 Sixth International Workshop on Managing Technical Debt*, MTD 2014, pp. 1–7, September 2014.

[11] Everton da S. Maldonado and Emad Shihab. Detecting and Quantifying Different Types of Self-Admitted Technical Debt. In *Proceedings of the 2015 IEEE 7th International Workshop on Managing Technical Debt*, MTD 2015, pp. 9–15, October 2015.

[12] Beat Fluri, Michael Wursch, and Harald C. Gall. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. In *Proceedings of the 14th Working Conference on Reverse Engineering*, WCRE 2007, pp. 70–79, October 2007.

謝辞

本研究を進めるにあたって，多くの方々にご支援，ご協力をいただきました．この場をお借りして深く謝意を表します．

主指導教員であり，本論文の審査委員を務めていただいた，奈良先端科学技術大学院大学先端科学技術研究科 松本健一教授には，本学で研究を行うにあたり，研究の進め方や発表の仕方など，丁寧にご指導，ご助言をいただきました．また，日々の研究だけではなく，多方面に渡り多大なご支援をいただきました．心より感謝申し上げます．

副指導教員である，奈良先端科学技術大学院大学先端科学技術研究科 飯田元教授には，学内での発表にて，多数のご質問とご指導をいただきました．心より感謝申し上げます．

副指導教員である，奈良先端科学技術大学院大学先端科学技術研究科 石尾隆准教授には，研究を進めるにあたり，数々のご指導，ご助言をいただきました．心より感謝申し上げます．

副指導教員であり，本研究を直接指導していただいた，奈良先端科学技術大学院大学先端科学技術研究科 畑秀明助教には，常日頃より熱心にご指導いただきました．心より感謝申し上げます．

和歌山大学システム工学部システム工学科 伊原彰紀講師には，研究や学生生活において数々のご指導をいただきました．心より感謝申し上げます．

奈良先端科学技術大学院大学先端科学技術研究科 Raula Gaikovina Kula 助教には，発表練習にて，多面的なご質問とご指導をいただきました．心より感謝申し上げます．

学生生活や研究会発表において，事務処理などの様々な面でサポートしていただいた高岸詔子氏に心より感謝申し上げます．高岸さんのお陰で，学生生活における事務手続きを大変スムーズに行うことができ，研究活動に専念することができました．

ソフトウェア工学研究室の学生の皆様には，発表練習や論文の添削にお付き合いいただくなど，多大なるご支援とご協力をいただきました．皆様のお陰で，非常に充実した 2 年間を過ごすことができました．心より感謝申し上げます．

関連発表論文

国内研究会・全国大会等

[1] 田内遥夏, 中才恵太朗, 畑秀明, 松本健一, “Waiting Self-Admitted Technical Debt の分析と考察,” 情報処理学会研究報告, ソフトウェア工学, volume 2018-SE-200, number 9, pages 1-6 2018 年 12 月.