ソースコードコメントに着目した不確かさとソフトウェア品質の関係調査

渡邉 紘矢
京都工芸繊維大学
工芸科学部 情報工学課程
h-watanabe@se.is.kit.ac.jp

崔 恩瀞
京都工芸繊維大学
情報工学・人間科学系
echoi@kit.ac.jp

水野 修
京都工芸繊維大学
情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

ソフトウェア開発中には開発者の知識不足により，既存プロジェクトのソースコードを十分に理解出来ないことがある．不確かさによるソースコードの複雑化は，ソフトウェアの品質や保守性に影響を与えている可能性があるが，この実証研究はまだ行われていない．そこで，本研究では，ソースコードコメントに存在する不確かさに着目し，不確かさによってソフトウェア品質にどのような影響があるかを調査した．具体的には，まず，ソースコードや設計に問題がある可能性を示す指標であるコードスメルと，ソースコードの品質メトリクスとして利用される循環的複雑度とクラス結合度を用いて，不確かさを表すキーワードをコメントに持つソースコードは品質が低いか調査した．調査の結果，不確かさはコードスメルの数への影響は見られないが，不確かさの影響を受けやすいコードスメルがあることがわかった．また，不確かさがあることで，*C/C++*，*Python* では循環的複雑度が高く，*C++*，*Python* ではクラス結合度が高くなることがわかった．

1. 緒言

ソフトウェア開発中には「不確かさ」と呼ばれる問題がある．不確かさの例としては，開発者の知識不足により，既存プロジェクトのソースコードを十分に理解出来ない，バグの修正方法など実装が正しいか分からないといった問題が挙げられる．機械学習などを用いたクラス分類器を例に挙げると，100%の正解率を達成することが難しく，誤った出力をする可能性があり，出力に不確かさが存在している．このような不確かさは，開発者にとって扱いにくいものであるため，開発者は直面した問題に対して一時的な措置をとる場合がある．この一時的な措置や不確かさが存在することによって，バグの混入や，ソースコードの複雑化を招く可能性がある．そのため，ソフトウェア工学において，不確かさを包容したソフトウェア開発は重要な研究課題の 1 つとして注目されている．

Git で管理されたプロジェクトを対象に不確かさを調査するツールを利用し，コミットメッセージに着目して不確かさを調査した研究がある [1–3]．このように，ソフトウェア開発において開発者が直面する不確かさについて実証研究があるが，その数は少ない．不確かさによってソースコードが複雑化することで，ソフトウェアの品質や保守性に影響を与えている可能性があるが，この実証研究はまだ行われていない．

そこで，本研究では，ソースコードコメントに着目し，不確かさによってソフトウェア品質にどのような影響があるかを調査する．具体的には，まず，ソースコードや設計に問題がある可能性を示す指標であるコードスメルに注目し，不確かさを表すキーワードをコメントに持つソースコードは多くのコードスメルが含まれているか調査する．次に，ソースコードの品質メトリクスとして利用される循環的複雑度とクラス結合度を用いて，不確かさを表すキーワードをコメントに持つソースコードは品質が低いか調査する．調査の結果，不確かさはコードスメルの数への影響は見られないが，不確かさの影響を受けやすいコードスメルがあることがわかった．また，不確かさがあることで，C/C++，Python では循環的複雑度が高く，C++，Python ではクラス結合度が高くなることがわかった．

2. 背景

2.1. コードスメル

技術的負債とは，ソフトウェア開発プロセスにおいて発生した問題を先延ばしにすることで，将来的に必要になるリファクタリング（ソフトウェアの外部から見た振る舞いを変えずに，内部構造を整理する作業 [4]）や修正のためのコストのことである．例えば，短い開発期間でリリースする必要があることから，発生した問題を一時的に修正するパッチを適用することで，適切な対応をとることを先延ばしにすることが挙げられる．これは，コードスメルの導入やバグの危険性の問題があるため，これらを検出し，解消していく必要がある [5].

コードスメルはコードや設計に問題がある可能性を示す指標として Fowler によって提案され，コードスメルが存在するソースコードはリファクタリングの実施が推奨されている [4]．数多く定義されているコードスメルの一例に過ぎないが，コードスメルは God Class，Feature Envy がよく知られている [4, 6]．God Class は他のクラスのことを知りすぎて巨大化したクラスであり，Feature Envy は他のクラスのデータを頻繁に参照していることを示している．これらは，巨大化したクラスを複数の小さなクラスへ分割することや，適切な他のクラスへメソッドを移動させることで，クラス間の依存関係を減らすリファクタリングを実施する必要がある．コードスメルを検出しリファクタリングすることで，ソフトウェアの品質と保守性の向上が期待できる．

ソフトウェア内に含まれるコードスメルを自動的に検査するツールの 1 つとして，SonarQube[1]がある．SonarQube は，事前に定義したルールベースのアルゴリズムに従ってコードスメルを検出するツールであり，ルールはプラグインとして追加できる機能を持っている [7].

2.2. 不確かさ

近年，不確かさを抱擁したソフトウェア開発は，ソフトウェア工学において重要な研究課題として注目されている [8]．不確かさの例としては，開発者の知識不足により，既存プロジェクトのソースコードを十分に理解出来ない，バグの修正方法など実装が正しいか分からないといった問題や，開発の初期段階に発生しやすい曖昧な要求，定まっていない設計など曖昧さによる問題が挙げられる．このような不確かさは，開発者にとって扱いにくいものであるため，開発者は直面した問題に対して一時的な措置をとる場合がある．この一時的な措置や不確かさが存在することにより，バグの混入や，ソースコードの複雑化を招く可能性がある．しかし，初めから不確かさのないソフトウェア開発を進めることは多くの労力と時間を必要とするため難しい.

ソフトウェア開発における不確かさは，Known Knowns，Known Unknowns，Unknown Unknown の 3 つのタイプに分けられる [9]．Known Knowns は不確かさが存在しない開発である．Known Unknowns は不確かさのある問題が存在する開発であり，開発者がその問題を認識している状態である．Known Unknowns における不確かさの問題は，Issue Tracking System やソースコード，仕様書内にメモとして記載し管理されている．Unknown Unknowns は不確かさのある問題が存在するが，開発者は何が不確かであるかを認識出来ていない状態である．

また，Prez-Palacin らは，不確かさを場所，レベル，性質の 3 つの観点で分類している [10]．場所は，どの部分に不確かさが現れているかを示し，モデルに含まれる部分と抽象化されている部分の境界を指すコンテキスト，そのモデル自体の構造，モデルに入力されるパラメータの 3 つに分類される．レベルは，不確かさを 5 段階で分類している．1 段階目は不確かさが欠落している状態 (Known Knowns に相当)，2 段階目は開発者は何らかの知識が欠落しているが，知識が欠落していることを認識している状態 (Known Unknowns に相当) である．3 段階目は開発者は知識が欠けていることを認識できていない状態 (Unknown Unknowns に相当)，4 段階目は自分が認識出来ていないことを認識するためのプロセスが欠落している状態，そして，5 段階目はメタな不確かさである．性質は，認識的と偶発的に分類される．認識的は，得られたデータが不十分である，データに対する理解度の欠落による不確かさ，偶発的は対象が持つランダム性による不確かさを表す．この分類における 3 段階目以上のレベルの不確かさについて，どのような問題があるか調査することが難しいため，既存の不確かさの研究 [1–3] では Known Unknowns のみを扱っている．

ソフトウェア開発において開発者が直面する不確かさについて実証研究があるが，その数は少ない．不確かさは，開発者にとって扱いにくいものであるため，不確かさを抱擁したソフトウェア開発を支援するツールを開発

[1] https://www.sonarqube.org/

する必要がある．実際のソフトウェアに含まれる不確かさを明らかにすることで，影響が大きく優先して解消する必要がある不確かさを指摘でき，この知見がツールの開発に役立つと考えられる．例えば，[1] ではコミットメッセージだけでなく，実際のソースコードや変更履歴と合わせた検証が必要であるとしている．また，コミットメッセージではなく，ソースコードコメントに着目した不確かさの調査はまだ行われていない．

2.3. 品質メトリクス

品質メトリクスとして，McCabe による循環的複雑度 [11]，Chidamber らによるクラス結合度が提案されている [12]．循環的複雑度はプログラムの実行経路の数を示している．例えば，if-else，switch-case などの条件分岐，for などの繰り返し文などが多く含まれることで，実行経路が増えるため，より複雑度が高い値を示すようになる．複雑度が高くなると，ソースコードのテストパターンの数が増えるため，テストコードの用意にかかる時間が増え，その保守性も低下することになる．

クラス結合度とは，他のクラスからメソッドが呼び出されている数，参照されたインスタンス変数の数を示している．つまり，クラス結合度は他のクラスやメソッドへの依存度の高さを示していると考えることができる．1 つのクラスやメソッドに対する仕様変更や修正が，複数のクラスやメソッドに影響するため，仕様変更や修正が困難になる．また，そのようなソースコードは理解も困難である．これらのメトリクスは，ソースコード品質へ与える影響を調査する目的で，広く利用されている [13]．

3. 調査概要

3.1. 調査目的

本研究の目的は，不確かさがあることで，ソフトウェアの品質にどのような影響が表れているか明らかにすることである．そこで，本研究ではソースコードコメントに着目したアプローチをとる．不確かさを表すキーワードを含むソースコードコメントが記述されているクラスやメソッドに着目し，不確かさがソフトウェア品質へ与える影響について調査した．

本研究の目的を達成するため，2 つの研究設問を設け，調査する．

RQ1 不確かさを表すキーワードをコメントに持つソースコードは多くのコードスメルが含まれているか？

RQ2 不確かさを表すキーワードをコメントに持つソースコードは品質が低いか？

3.2. 調査対象のデータセット

本論文で調査対象としてコード片とソースコードコメントの対のデータセットである，Source Code Analysis Dataset（以降 SCAD）を選択した [14]．このデータセットは，ソースコードコメントの予測や自然言語を用いたソースコードコメントの生成などの研究への活用を目的に作成された．

このデータセットがどのように作成されたのかを説明する．まず GitHub[2]から，再配布可能なライセンスを持ち，10 個以上のスターを持つ 108,568 個のオープンソースソフトウェア (OSS) プロジェクトを取得する．取得の対象としているプログラミング言語は，Java，C 言語，C++，Python である．取得したソースコードから，Doxygen[3]を使用してコード片とソースコードコメントの対を抽出する．コード片は，クラス，メソッド，関数，変数宣言の粒度で抽出される．Doxygen は 106,304 個（108,568 個中）の OSS プロジェクトで正常に実行され，合計 16,115,540 件のコード片とソースコードコメントの対が得られた．この対応関係がデータセットとして，提供されている．

4. RQ1 不確かさを表すキーワードをコメントに持つソースコードは多くのコードスメルが含まれているか？

4.1. 動機

不確かさを持つソースコードには，不確かさを持たないソースコードよりもコードスメルが多い可能性がある．プログラミング言語はそれぞれ，深層学習や Web 開発，アプリ開発といった得意な開発分野を持ち，習慣的な設計方法やプログラムの書き方が異なる場合がある．そのため，プログラミング言語によって，不確かさの影響は異なると考えられる．不確かさを持つソースコードとコードスメルの関係が明らかになることで，開発者が

[2]https://github.com/
[3]https://www.doxygen.nl/index.html

感じる不確かさは，リファクタリングの実施，設計の見直しが必要であることの指標になると考えられる．

4.2. 方法

RQ1 に答えるために，不確かさを表すキーワードをコメントに持つソースコードのコードスメルの数と不確かさを表すキーワードをコメントに持たないソースコードのコードスメルの数を比較した．調査方法は，以下の手順で進めた．この手順を図 1 に示す．

手順 1-1 データセットを不確かさを含むもの，含まないものに分類する．

手順 1-2 プログラミング言語ごとに適切なサンプルサイズでデータをサンプルする．

手順 1-3 SonarQube を使ってコードスメルを検出する。

手順 1-4 マンホイットニーの U 検定で有意差検定する。

手順 1-1 では，3.2 節で述べた SCAD の全てのソースコードコメントを不確かさを持つものと持たないものの 2 クラスに分類する．不確かさがあるという定義は，本論文では，ソースコードコメントに不確かさを表すキーワードを持ち，かつ少なくとも 1 個の特徴単語を含む，とする．本論文で利用する不確かさを表すキーワード，およびそれに対応する特徴単語を表 1 に示す．これは，村本らの研究 [1] で，頻出度が高い 20 個の不確かさを表すキーワードとして報告されたものである．

手順 1-2 では，SCAD に含まれているデータをプログラミング言語ごとに適切なサンプルサイズでサンプルする．適切なサンプルサイズは，信頼水準 95%，許容誤差 5%としてサンプルサイズを計算した．この時，取り出すのは不確かさを表すコメントに対する関数やクラスなどのスニペットだけでなく，関数やクラスが含まれるファイル全体を取得する．手順 1-2 でファイル全体を取得することにしたのは，コードスメルの検出に SonarQube を利用しているためである．一般的な SonarQube の利用方法は，プロジェクト全体やファイル群を入力とする．そのため，関数やクラスなどのコード片を SonarQube に入力した場合，構文解析に失敗する場合がある．構文解析に失敗した場合，SonarQube ではコードスメルの検出が出来ない．これを避けるため，今回はファイル全体を取り出すことにした．

手順 1-3 では，2.1 節で述べた SonarQube を用いてコードスメルを検出する．SonarQube がコードスメルの検出に用いるルールは SonarQube Community Edition Version 9.0 で予め導入されているものを利用する．ただし，Java に対する検査では，予め導入されているルールに加え，Antipatterns-CodeSmell プラグイン[4]によるルールを追加している．これは既存のツールで検出できるコードスメルより，クラスごと，クラスメソッドごとなど検出粒度に分けて調査するために，既存のルールでは十分なコードスメルを検出出来ないと考えられているためである．また，このルールのうち，ルールにつけられた重要度が高いもののみ検出する．具体的には，Blocker，Critical，Major の 3 つの重要度のルールを検出し，Minor，Info の 2 つの重要度の低いルールは検出しない．これは，本質的ではないコードスメルが過剰に検出されてしまうことを避けるためである．

一般的に，コードスメルの数はファイル行数が多いほど多くなる．このようなファイル行数の影響を考慮するために，ファイル 1 行あたりのコードスメルの数を求め比較することにする．

手順 1-4 では不確かさを持つクラス群と，不確かさ持たないクラス群との間で，コードスメルの数の差に対してマン・ホイットニーの U 検定の片側検定を行う．また，有意差以外の指標としてノンパラメトリックな効果量測定法である Cliff の $|\delta|$ を利用する [15]．Cliff の $|\delta|$ では，測定された差の大きさを推定することが出来る．Cliff の $|\delta|$ が示す効果量の大きさは，$0.147 < |\delta|$ で無視できる，$0.147 \leq |\delta| < 0.330$ で小，$0.330 \leq |\delta| < 0.474$ で中，$0.474 \leq |\delta|$ で大となる．

4.3. 結果

手順 1-1 で不確かさを表すキーワードとその特徴単語を用いて，不確かさを持つデータと不確かさを持たないデータに分けた．この結果，各キーワードに対して一致したデータ数を表 2 に示す． may が 106,680 個，might が 20,968 個，unknown が 6,565 個検出され，その合計が 145,807 個となる．不確かさを含むデータとして分類されたデータの合計は 155,560 個であることから，3 つのキーワードが占める割合は 93.7%となる．

SCAD 全体とプログラミング言語ごとに対して，コードスメルの数の統計量を表 3，表 4 に示す．また，マン・

[4]https://github.com/davidetaibi/sonarqube-anti-patterns-code-smells

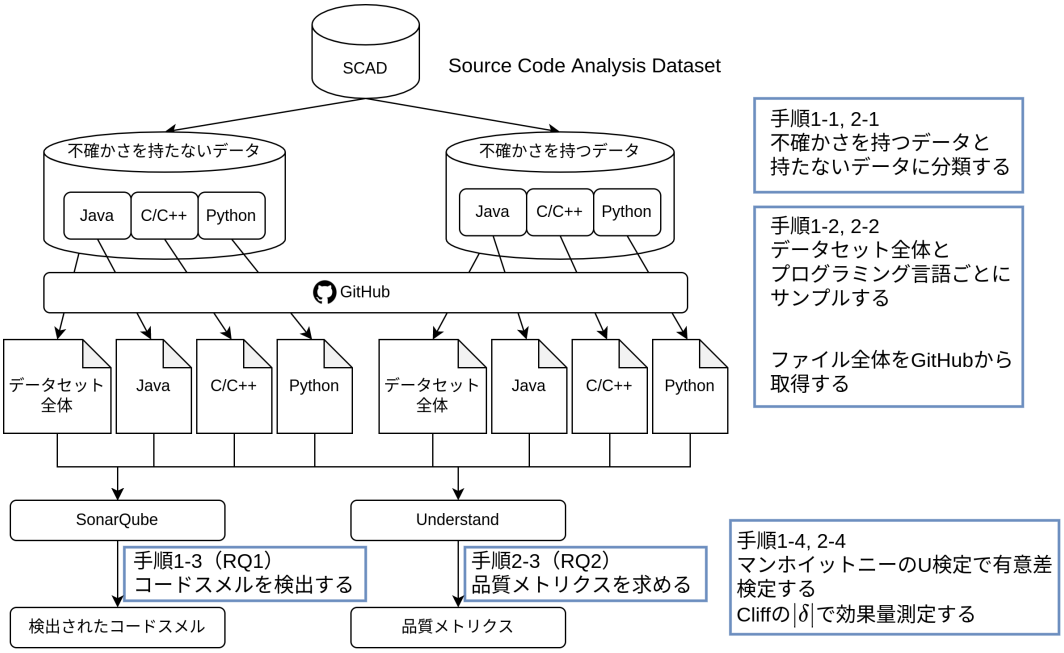

図 1. 調査方法の概要図

表 1. 不確かさを表すキーワードおよびそれに対応する特徴単語 [1]

不確かさを表すキーワード	特徴単語
ambiguous	call, valu, type, case, renam, avoid, name, differ, one, warn, error
arcane	trigger, damag, stack, build, make, take, shot, entri, seem, empti, column, will, error, problem
changeable	field, set, cach, default, make, offset, disk, select, pass, check, addit, start, found, error
debatable	case, system, getput, seem, name, place, want, think, realli
dubious	valu, code, bit, case, replac, get, seem, use, remov, avoid, warn, error
doubtful	see, realli, one, work, want, look
erratic	enabl, issu, behavior, result, logic, workaround, handl, appli, arm, caus, affect, part, bug
fuzzy	option, translat, way
irregular	break, switch, pass, valu, part, usag, approach, place, help, condit, work
may	case, mayb, caus, need, want, sinc, contain, time, differ, happen, mean
might	case, tri, get, want, need, caus, differ, lock, happen
obscure	code, case, problem, result, need, get, call, caus, bug, error
probably	get, need, want, caus, work, doesnt, way
risky	secur, case, data, run, potenti, reduc, dont, avoid, need
tentative	method, implement, problem, call, tentat, support, definit, work, like, allow, way, done, bug
unclear	valu, seem, name, reason, user, differ, like
unknown	type, messag, return, fail, case, tri, ignor, handl, reason, name, report, caus, will, error, warn
unreliable	test, case, detect, run, tri, set, remov, check, chang, time, dont, result, need
unsure	case, seem, work, dont, need, think, result
vague	return, line, function, set, get, attempt, document, time, think, caus, error

ホイットニーの U 検定と効果量として Cliff の $|\delta|$ の結果を表 5 に示す.

検定結果より，データセット全体や，どのプログラミング言語においても有意差はなく，その効果量も無視できる程度であった.

コードスメルの検出ルールについて，頻出度の高い上位 10 件を比較したところ，Java は 7 件，C/C++は 10 件全て，Python は 8 件一致していた．不確かさを持つかどうかによらず，出現頻度の高い検出ルールの項目は似た傾向にあることがわかる．つまり，不確かさを表すキー

表 2. 調査対象のデータセットに含まれるソースコードコメントのうち不確かさを表すキーワードを含むソースコードコメントの数

不確かさを表すキーワード	ソースコードコメントの数
ambiguous	1,375
arcane	48
changeable	146
debatable	25
dubious	6
doubtful	38
erratic	12
fuzzy	247
irregular	202
may	106,680
might	20,968
obscure	156
probably	6,565
risky	65
tentative	83
unclear	157
unknown	18,159
unreliable	381
unsure	143
vague	104
合計	155,560

表 3. 不確かさを持つソースコードに対する 1 行あたりに含まれるコードスメルの数の統計量

	SCAD 全体	Java	C/C++	Python
平均	0.048	0.072	0.029	0.032
第 1 四分位数	0.013	0.036	0.005	0.010
中央値	0.035	0.059	0.012	0.020
第 3 四分位数	0.064	0.088	0.028	0.038

表 4. 不確かさを持たないソースコードに対する 1 行あたりに含まれるコードスメルの数の統計量

	SCAD 全体	Java	C/C++	Python
平均	0.049	0.079	0.041	0.034
第 1 四分位数	0.013	0.035	0.006	0.013
中央値	0.030	0.063	0.013	0.024
第 3 四分位数	0.062	0.097	0.034	0.041

ワードをコメントに持つソースコードのコードスメルの内容と不確かさを表すキーワードをコメントに持たないソースコードのコードスメルの内容の間に，差異は無いことが考えられる．頻出度上位 10 件のうち，片方のみ

表 5. 1 行あたりに含まれるコードスメルの数に対するマン・ホイットニーの U 検定の結果と Cliff の $|\delta|$ による効果量

対象データ	有意差	効果量 $\|\delta\|$
SCAD 全体		0.088
Java		0.037
C/C++		0.049
Python		0.088

表れている検出ルールの一覧を以下に示す．C/C++においては，片方のみ表れている検出ルールは無かった．

Java

(1) パッケージの宣言はソースファイルのディレクトリと一致させるべきである

(2) コードの一部をコメントアウトするべきではない

(3) インクリメント (++) およびデクリメント (--) 演算子は，メソッド呼び出しの中で使用したり，式の中で他の演算子と混ぜて使用するべきではない

(4) 例外ハンドラは元の例外を維持するべきである

Python

(1) SystemExit を再度 raise すべきである

(2) 制御フロー文「if」「for」「while」「switch」「try」は深くネストさせるべきではない

RQ1 に対する回答を以下にまとめる．

> 不確かさを表すキーワードをコメントに持つ場合と，持たない場合のコードスメルの数は同程度である．

5. RQ2 不確かさを表すキーワードをコメントに持つソースコードは品質が低いか？

5.1. 動機

不確かさによるソフトウェア品質への影響を明らかにすることで，事前にある一定のソフトウェア品質を確保するためには，どの程度の不確かさを許容できるのか見積もることができる．

5.2. 方法

RQ2 に答えるために，不確かさを表すキーワードをコメントに持つソースコードの品質メトリクスの値と，不確かさを表すキーワードをコメントに持たないソースコードの品質メトリクスの値を比較した．調査方法は，以下の手順で進めた．この手順を図 1 に示す．

手順 2-1 データセットを不確かさを含むもの，含まないものに分類する．

手順 2-2 プログラミング言語ごとに適切なサンプルサイズでデータをサンプルする．

手順 2-3 品質メトリクスを求める．

手順 2-4 マンホイットニーの U 検定で有意差検定する．

手順 2-1~ 手順 2-2 は，RQ1 の手順 1-1，手順 1-2 と同様であるため，本節では省略する．手順 2-2 でのサンプルサイズは，信頼水準 95%，許容誤差 5%として計算した．

手順 2-3 では，2.3 節で述べた循環的複雑度，クラス結合度の 2 つの品質メトリクスを求める．品質メトリクスを求めるため，ソースコード解析ツールである Understand 6.0[5]を利用した．手順 2-2 では，ファイル全体を取得することから，これらの品質メトリクスはファイル全体に対して計算される．また，循環的複雑度，クラス結合度のメトリクスは，ファイル行数が多いほど，循環的複雑度，クラス結合度が高い値を示しやすくなる．このようなファイル行数の影響を考慮するために，ファイルの 1 行あたりの品質メトリクスの値を求め，比較することにする．

循環的複雑度は，メソッド単位で計算されるめ，そのファイル内に含まれる全てメソッドの循環的複雑度の合計を求める．この合計値を，ファイル行数で割ったものを 1 行あたりの循環的複雑度とする．クラス結合度は，クラスから内部クラス，匿名クラスを分離し，それぞれに対してクラス結合度を求める．また，ファイル内にあるインターフェースや Enum クラスに対しても求められるため，それら全ての値の合計を求める．この合計値をファイル行数で割ったものを 1 行あたりのクラス結合度とする．循環的複雑度，クラス結合度の値はどちらも非負であり，非有界である．値は大きくなるほどより複雑であり，より結合していることを示す．

[5]http://www.techmatrix.co.jp/product/understand/index.html

表 6. 不確かさを持つソースコードに対する 1 行あたりの循環的複雑度の統計量

	SCAD 全体	Java	C/C++	Python
平均	0.201	0.191	0.127	0.245
第 1 四分位数	0.151	0.158	0.000	0.202
中央値	0.212	0.200	0.143	0.251
第 3 四分位数	0.264	0.234	0.195	0.292

表 7. 不確かさを持たないソースコードに対する 1 行あたりの循環的複雑度の統計量

	SCAD 全体	Java	C/C++	Python
平均	0.174	0.187	0.116	0.232
第 1 四分位数	0.096	0.149	0.000	0.177
中央値	0.178	0.190	0.083	0.231
第 3 四分位数	0.239	0.228	0.181	0.285

表 8. 不確かさを持つソースコードに対する 1 行あたりのクラス結合度の統計量

	SCAD 全体	Java	C++	Python
平均	0.062	0.084	0.075	0.042
第 1 四分位数	0.024	0.036	0.023	0.019
中央値	0.045	0.067	0.050	0.036
第 3 四分位数	0.734	0.110	0.094	0.055

手順 2-4 では不確かさを含むクラス群と，不確かさを含まないクラス群との間で，品質メトリクスの値の差に対してマン・ホイットニーの U 検定の片側検定を行う．また，RQ1 同様 Cliff の $|\delta|$ を利用する [15].

5.3. 結果

SCAD 全体やプログラミング言語ごとの，循環的複雑度の統計量を表 6，表 7 に示す．クラス結合度の統計量を表 8，表 9 に示す．また，マン・ホイットニーの U 検定と Cliff の $|\delta|$ の結果を表 10，表 11 に示す．

チェックマークは有意差 ($p < 0.05$) を示しており，効果量は Cliff の $|\delta|$ の値を記載している．循環的複雑度は，SCAD 全体，C/C++，Python のみ有意差があり，その効果量として C/C++は無視できる程度，Python は小程度であった．クラス結合度は，C++，Python のみ有意差があり，その効果量として C/C++は小程度，Python は無視できる程度であった．Java に対するクラス結合度は有

表 9. 不確かさを持たないソースコードに対する 1 行あたりのクラス結合度の統計量

	SCAD 全体	Java	C++	Python
平均	0.083	0.115	0.065	0.036
第 1 四分位数	0.017	0.048	0.000	0.011
中央値	0.049	0.087	0.027	0.027
第 3 四分位数	0.107	0.149	0.089	0.048

表 10. 1 行あたりの循環的複雑度に対するマン・ホイットニーの U 検定の結果と Cliff の $|\delta|$ による効果量

対象データ	有意差	効果量 $\|\delta\|$
SCAD 全体	✓	0.190
Java		0.041
C/C++	✓	0.140
Python	✓	0.190

表 11. 1 行あたりのクラス結合度に対するマン・ホイットニーの U 検定の結果と Cliff の $|\delta|$ による効果量

対象データ	有意差	効果量 $\|\delta\|$
SCAD 全体		0.047
Java		0.192
C++	✓	0.154
Python	✓	0.128

意差が出ていないが，その効果量は小程度となっている．これは，Java の場合は不確かさを持つソースコードの方が，クラス結合度は高くなることを示している．追加の調査として，Java を対象に不確かさを持たないソースコードの方がクラス結合度は高くなるか，マン・ホイットニーの U 検定を片側検定で行った．この片側検定では有意差がある結果となり，効果量測定より効果量は小程度であることを示した．RQ2 に対する回答を以下にまとめる．

不確かさを表すキーワードをコメントに持つ方が，C/C++，Python に対して循環的複雑度は高く，C++，Python に対してクラス結合度はより高くなる．Java に対するクラス結合度では，不確かさを表すキーワードをコメントに持たない方がクラス結合度は高くなる．

6. 議論

6.1. RQ1 不確かさを表すキーワードをコメントに持つソースコードは多くのコードスメルが含まれているか？

不確かさを持つソースコードのコードスメルの数と不確かさを持たないソースコードのコードスメルの数をファイル単位で比較した．その結果，全てのプログラミング言語において有意差は見られなかった．コードスメルは，ソースコードの設計上の問題を示す指標である．つまり，ソフトウェア開発において不確かさがあったとしても，ソースコードの設計上の問題への影響は無いことが考えられる．開発中に感じる不確かさは，リファクタリングが必要であること以外に，その不確かさの原因となる他の要素があると考えられる．本論文では，ファイル単位で調査したため，関数やメソッド単位での調査に比べると粒度が大きく，影響が表れにくくなっている可能性がある．そのため，今後は関数やメソッド単位のより細かい粒度で調査し，その影響について明らかにする必要があると考えられる．

不確かさを持つ場合，もしくは不確かさを持たない場合の片方のみ表れている検出ルールがあった．Java の (1) の検出ルールは，コードスメルを解析する際に，ファイルを配置したパスやそのソースファイルのファイル名を変更したため，表れたルールである．そのため，重要なルールではないと考えられる．Java の (2)，(3) に示す検出ルール，Python の (2) に示す検出ルールは可読性へ悪影響を示唆している．Java の (4) に示す検出ルール，Python の (1) に示すルールは例外を無視するなど，エラーハンドリングにおける適切な対応が行われていないことを示している．これらは，開発者がコードを理解しようとする妨げになる可能性がある．そのため，不確かさの原因となるコードスメルである，または，不確かさによる影響を示しているコードスメルであることを示唆

していると考えられる．

6.2. RQ2 不確かさを表すキーワードをコメントに持つソースコードは品質が低いか？

C/C++，Python における循環的複雑度，C++，Python におけるクラス結合度では，ともに有意差が見られ，Java では循環的複雑度，クラス結合度ともに有意差が見られなかった．これは，不確かさによる悪影響を受けるかどうかはプログラミング言語ごとに異なるが，少なくとも C/C++と Python においてソースコードの品質に悪影響を与えている可能性があることを示している．また，ソフトウェア開発において良い品質を保つために，不確かさを解消する必要性があることを実証的に示せたと考えられる．

しかし，初めから一切不確かさを含まないように開発を進めることは難しく，ある程度の不確かさが含まれることを許容する必要がある．そのためには，どの程度の不確かさが含まれていた場合に，品質がどの程度悪くなるか関係を調べる必要があると考えられる．不確かさの程度と品質の関係が明らかになることで，予めソフトウェアの品質を見積もることが可能になるからである．

Java のクラス結合度について，不確かさを持たない方がクラス結合度は有意に高くなる結果が得られた．これは，不確かさによる品質への悪影響として予想していた結果と逆の結果である．また，この結果は，Java を利用したソフトウェア開発において，不確かさを解消した設計はクラス結合度が高くなってしまう可能性があると考えられる．例えば，用いられている設計モデルやフレームワークといったプロジェクトの開発背景による影響があると考えられる．

7. 妥当性への脅威

不確かさの識別には，不確かさを表すキーワードや，そのキーワードと同時に現れる特徴単語を利用し，正規表現を用いてソースコードコメントと照合することで，機械的に不確かさを識別した．この識別方法では，先行研究によって，不確かさを表すキーワードには不確かさの特徴的な傾向がみられ，特徴単語は不確かさの内容を表していると推測出来ることが報告されている．しかし，不確かさを表すキーワードがソースコードコメントに含まれていたとしても，実際に不確かさがあるとは限らない．同様に，不確かさを表すキーワードを含んでいなかったとしても，不確かさがある可能性がある．このような，偽陽性や偽陰性の存在による影響がある．その影響については，特定したソースコードコメントに対して，目視分類する必要がある．

また，不確かさを表すキーワードや特徴単語は，共にコミットメッセージ内に書かれたものを対象として報告されたものである．今回はソースコードコメントを対象に適用することで調査した．コミットメッセージやソースコードコメントは，どちらも自然言語を用いて記述されているため，その傾向に差は無いと考えられる．

8. 関連研究

Famelis らは，不確かさの存在を表現するモデルとして，Partial Model を利用したアプローチを提案している [16]．深町らは，この Partial Model を利用した提案に基づいて，Java プログラミング環境を提案している [17]．また，Esfahani らは，システム開発の初期段階で，不確かさがある状況下でシステムアーキテクチャを設計するためのフレームワークを提案している [18]．不確かさの影響を実証的に調査して得られた知見は，不確かさを管理し，ソフトウェア開発を支援するツールの開発に役立つと考えられる．不確かさに関する実証研究のため，Git で管理されたプロジェクトを対象に不確かさを調査するためのツールが提案されている [19]．このツールを利用し，コミットメッセージに着目して不確かさを調査した研究がある [1–3]．本研究では，ソースコードコメントに着目し，不確かさとソフトウェア品質の関係について調査した．

9. 結言

本研究では，ソースコードコメントに着目し，不確かさがあることで，ソフトウェア品質にどのような影響が表れているか調査した．調査対象は Java，C 言語，C++，Python の 4 つのプログラミング言語である．

調査の結果，4 つのプログラミング言語に対しては，不確かさを持つソースコードのコードスメルの数と不確かさを持たないソースコードのコードスメルの数には有意な差は無かった．品質メトリクスに対しては，不確かさがあることで，C/C++，Python では循環的複雑度が有意に高く，C++，Python ではクラス結合度が有意に高く

なった．一方，Java では不確かさを持たない方が，クラス結合度は有意に高くなる結果となり，予想していた結果とは逆の結果が得られた．

プログラミング言語によって不確かさの影響が異なることが明らかとなったが，プログラミング言語のどの要素が影響しているかは明らかに出来ていない．どのようなプログラム，どのような機能を利用すれば良いかについては，言語仕様の観点から調査し，明らかにする必要がある．

参考文献

[1] 村本大起，江　冠達，村岡北斗，深町拓也，鵜林尚靖，亀井靖高，佐藤亮介，“ソフトウェア開発における不確かさに着目した OSS コミットログ解析，” ソフトウェアエンジニアリングシンポジウム 2017 論文集，vol.2017，pp.122–129，2017.

[2] 村岡北斗，鵜林尚靖，亀井靖高，佐藤亮介，“Revert に着目した不確かさに関する実証的分析，” ソフトウェアエンジニアリングシンポジウム 2019 論文集，vol.2019，pp.31–40，2019.

[3] N. Ubayashi, Y. Kamei, and R. Sato, “When and Why Do Software Developers Face Uncertainty?,” Proc. of QRS, pp.288–299, 2019.

[4] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

[5] W. Cunningham, “The WyCash portfolio management system,” Proc. of OOPSLA, pp.29–30, 1992.

[6] W.H. Brown, R.C. Malveau, H.W. McCormick, and T.J. Mowbray, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, 1998.

[7] V. Lenarduzzi, A. Sillitti, and D. Taibi, “A Survey on Code Analysis Tools for Software Maintenance Prediction,” Proc. of ICSE, pp.165–175, 2018.

[8] D. Garlan, “Software engineering in an uncertain world,” Proc. of FoSER, p.125–128, 2010.

[9] S. Elbaum and D.S. Rosenblum, “Known unknowns: Testing in the presence of uncertainty,” Proceedings of the 22nd International Symposium on Foundations of Software Engineering (FSE), pp.833–836, 2014.

[10] D. Perez-Palacin and R. Mirandola, “Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation,” Proc. of ICPE, pp.3–14, 2014.

[11] T.J. McCabe, “A Complexity Measure,” IEEE Transactions on Software Engineering, vol.SE-2, no.4, pp.308–320, 1976.

[12] S.R. Chidamber, D. Darcy, and C. Kemerer, “Managerial use of metrics for object-oriented software: An exploratory analysis,” IEEE Transactions on Software Engineering, vol.24, pp.629–639, 1998.

[13] G. Bavota and B. Russo, “A large-scale empirical study on self-admitted technical debt,” Proc. of MSR, pp.315–326, 2016.

[14] B. Gelman, B. Obayomi, J. Moore, and D. Slater, “Source Code Analysis Dataset,” Data in Brief, vol.27, pp.1–6, 2019.

[15] G. Macbeth, E. Razumiejczyk, and R. Ledesma, “Cliff’s Delta Calculator: A non-parametric effect size program for two groups of observations,” Universitas Psychologica, vol.10, pp.545–555, 2011.

[16] M. Famelis, R. Salay, and M. Chechik, “Partial models: Towards modeling and reasoning with uncertainty,” Proc. of ICSE, pp.573–583, 2012.

[17] 深町拓也，鵜林尚靖，細合晋太郎，亀井靖高，“不確かさを包容する Java プログラミング環境，” 情報処理学会研究報告，vol.2015-SE-187，no.21，pp.1–8，2015.

[18] N. Esfahani, K. Razavi, and S. Malek, “Dealing with uncertainty in early software architecture,” Proc. of FSE, pp.1–4, 2012.

[19] 村岡北斗，村本大起，鵜林尚靖，亀井靖高，佐藤亮介，“Git 開発履歴情報に基づく不確かさの可視化，” 情報処理学会研究報告，vol.2017-SE-196，no.29，pp.1–8，2017.