卒 業 研 究 報 告 書

題　目　　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

指導教員　　水野　修　教授

崔　恩瀞　助教

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　18122508

氏　　名　　渡邉　紘矢

令和4年2月10日提出

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

令和 4 年 2 月 10 日　　18122508　渡邉　紘矢

概　要

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ある．また，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も挙げられる．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り，その結果バグの混入や，ソースコードの煩雑化を招く可能性がある．そのため，近年，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として注目されている．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．例えば，不確かさによるソースコードの煩雑化は，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した．次に，ソースコードの品質メトリクスとして利用される複雑度と結合度を用いて，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 では複雑度が高く，C++，Python では結合度が高くなることがわかった．一方，Java では不確かさを持たない方が，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．この結果は，ソフトウェア品質が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によって異なることを示唆している．

目　次

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．この一時的な措置などにより，バグの混入や，ソースコードの煩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．

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ツールが提案されている [1]．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 [2, 3, 4]．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煩雑化することで，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．

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した．次に，ソースコードの品質メトリクスとして利用される複雑度と結合度を用いて，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．調査には，ソースコードコメントの分析のために作成されたデータセットを用いる．このデータセットは Java，C 言語，C++，Python の 4 つのプログラミング言語のコード片とソースコードコメントの情報を持っている．そのため，各プログラミング言語ごと調査し，その特徴を明らかにした．

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 では複雑度が高く，C++，Python では結合度が高くなること

がわかった．一方，Java では不確かさを持たない方が，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．この結果は，ソフトウェア品質が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によって異なることを示唆している．

最後に本論文の章構成を示す．まず，第 2 章では本研究で必要となる背景知識について述べる．第 3 章では，本研究の調査目的について，調査対象とするデータセットの概要について述べる．第 4 章では，1 つ目の研究設問における調査目的，調査方法や調査結果について述べる．第 5 章では，2 つ目の研究設問における調査目的，調査方法や調査結果について述べる．第 6 章では調査結果について議論する．第 7 章では妥当性への脅威について説明し，最後に第 8 章で結言を述べる．

2. 背景

2.1 技術的負債

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 [5]）や修正のためのコストのことである．例えば，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．先延ばしされる理由として，短い開発期間でリリースする必要があることや，そのために開発速度を向上する必要があることが挙げられる．これは，2.2 章で述べる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 [6].

2.2 コードスメル

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 によって提案された [5]．また，コードスメルは技術的負債の 1 つとして注目され，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．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 [5, 7]．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 (注 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 [8]．また検出されたコードスメルに対して，リファクタリングするべきソースコードを開発者に提示するツールとして，Moha らによる DECOR

(注 1)：https://www.sonarqube.org/

や，PMD (注 2)といったツールが数多く開発されている [9, 10].

2.3 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 [11]．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．この一時的な措置などにより，バグの混入や，ソースコードの煩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．

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 [12]．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 [13]．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 [14]．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 [1]．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 [2, 3, 4].

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 [15]．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．つまり，Unknown Unknowns の状態は，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

(注 2)：https://pmd.github.io/

研究 [2, 3, 4] 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，[2] 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．

2.4 品質メトリクス

品質メトリクスには，複雑度，結合度を示すメトリクスがある．複雑度には，McCabe による循環的複雑度が提案されている [16]．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．結合度は，Chidamber らによる他のクラスとの結合数が提案されている [17]．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が増えると結合度は高くなる．つまり，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 [18]．

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 [19]．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．また，深層学習を用いたソースコードの脆弱性解析の研究に利用できるデータセットとして取り上げられている [20].

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 (注 3)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 (注 4)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

(注 3)：https://github.com/
(注 4)：https://www.doxygen.nl/index.html

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

ソースコードへの影響として，ソースコードの設計上の問題を示す指標であるコードスメルに注目する．不確かさがソースコードに悪影響を与えると仮定すると，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 を答えるために，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 4.1 に示す.

手順 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　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 3　SonarQube を使ってコードスメルを検出する。
手順 4　マンホイットニーの U 検定で有意差検定する。

手順 1 では，3.2 章で述べた 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．データセットには，Java，C 言語，C++，Python の 4 種類のソースコードが含まれているため，これら 4 種類を調査対象とする．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

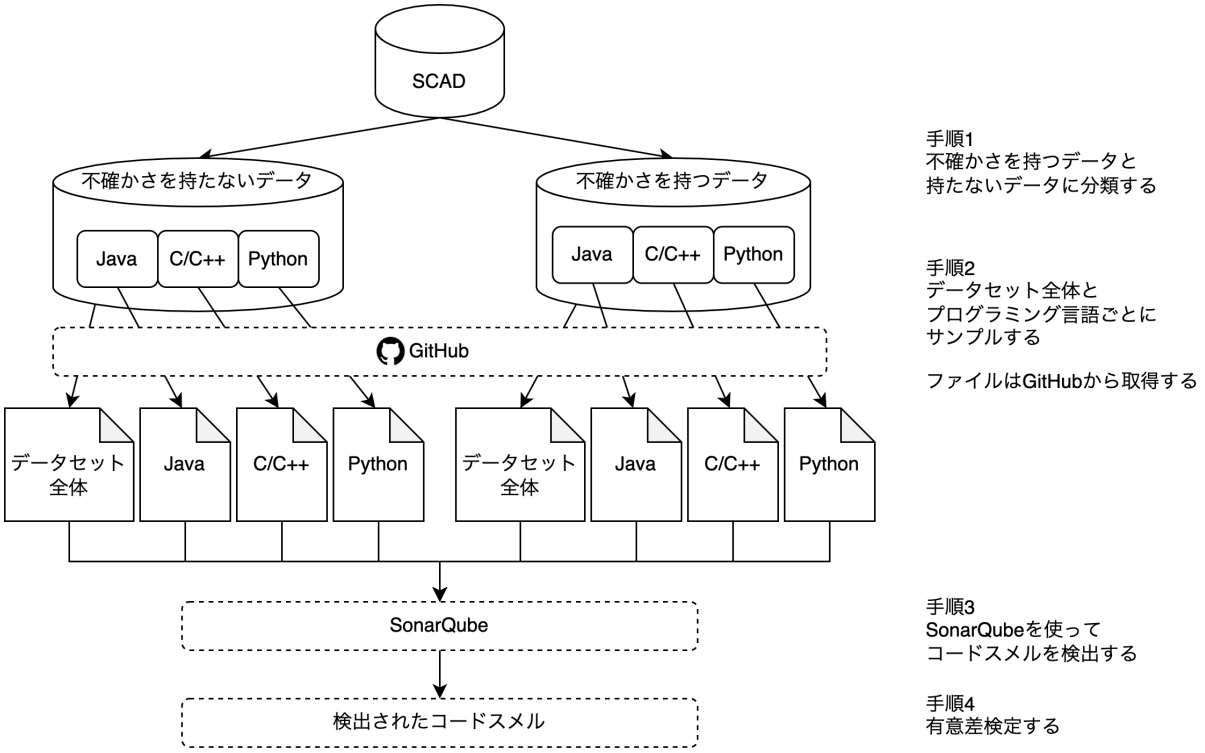


図 4.1　RQ1 における調査方法の概要図

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場合，不確かさを持つという定義を用いる．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 4.1 に示す．これは，村本らの研究 [2] 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 2 では，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．データセット全体を分析することは時間がかかってしまうため，サンプルしたデータに対して分析することにした．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．SCAD では，関数やクラスのコード片が含まれているファイル名，GitHub リポジトリへの URL，コミットハッシュを取得できる．この情報を用いて，GitHub から該当するファイルを取得することが出来る．

手順 3 では，2.2 章で述べた 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 (注 5) 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．例えば,「テストコードがない」「ファイル名は次の正規表現に一致するようファイル名を変更する必要がある」といったルールが挙げられる.「テストコードがない」については，今回はプロジェクト全体ではなく，ファイル単体での調査であるため，テストコードは必ず存在しない.「ファイル名は次の正規表現に一致するようファイル名を変更する必要がある」は，調査の前処理としてファイル名を変更し，ファイル名が重複しないようにしているためである．これらのルールは Minor，もしくは Info の重要度が設定されていることが多い．そのた

(注 5)：https://github.com/davidetaibi/sonarqube-anti-patterns-code-smells

め，Minor，Info の重要度を持つルールを検出しないことで，本質的ではないルールを一括して除外することにした．

手順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．

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 4 では，マン・ホイットニーの U 検定を用いて有意差を比較する．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，コードスメルの数に有意差をマン・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 [21]．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．これを表 4.2 に示す．

表 4.1　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

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

前ページからの続き

不確かさを表すキーワード	特徴単語
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

表 4.2　Cliff の $|\delta|$ の効果量の大きさ

Cliff の $\|\delta\|$ の値	効果量の大きさ
$\|\delta\| < 0.147$	無視できる程度
$0.147 \leq \|\delta\| < 0.330$	小程度
$0.330 \leq \|\delta\| < 0.474$	中程度
$0.474 \leq \|\delta\|$	大程度

4.3 結果

手順 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 4.3 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．

データセット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 4.4，表 4.5 に示す．統計量をもとに作成した箱ひげ図を図 4.2 に示す．図の三角マーカーは平均値を示している．また，マン・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 4.6 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．

コードスメル検出ルールのうち，頻出度の上位 10 件のルールの一覧を表 4.7~表 4.14 に示す．データセット全体と各プログラミング言語に対して，不確かさを持つ場合と持たない場合のコードスメルの検出ルールをそれぞれ示している．

表 4.3　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 4.4　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

	データセット全体	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

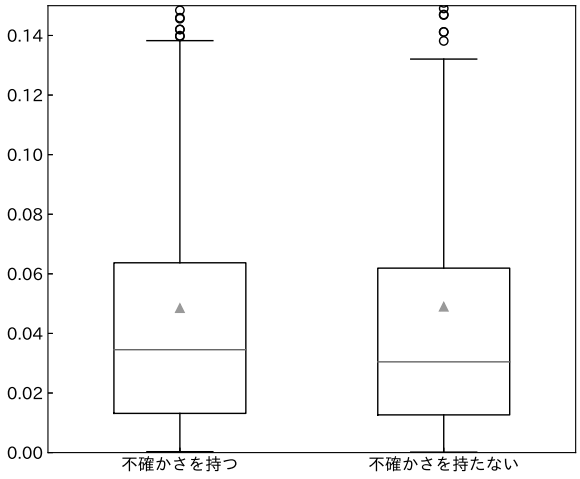
表 4.5　不確かさを持たないソースコードに対する 1 行あたりに含まれるコードスメルの数の統計量

	データセット全体	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

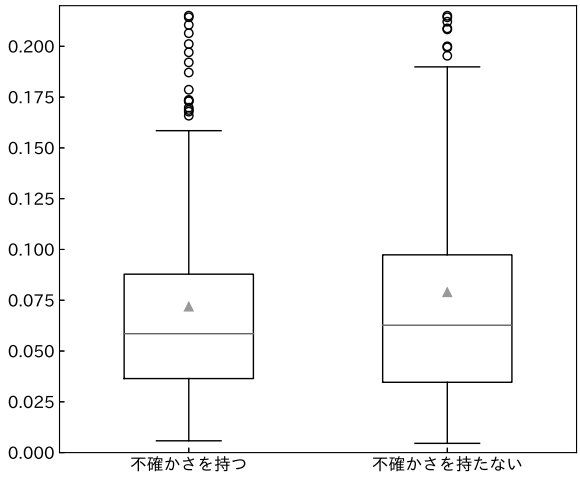
表 4.6　1 行あたりに含まれるコードスメルの数に対するマン・ホイットニーの U 検定の結果と Cliff の $|\delta|$ による効果量

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

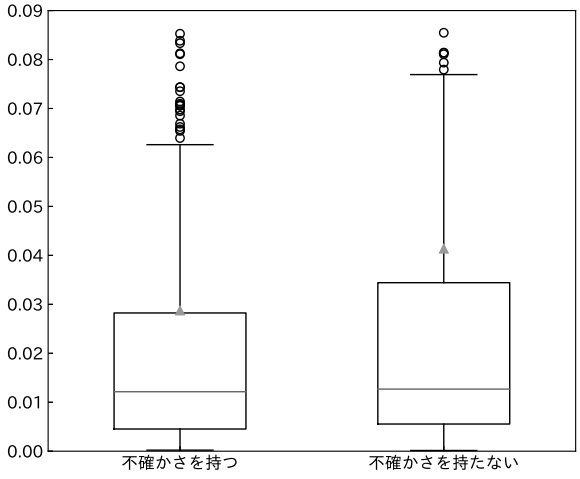
効果量の大きさ：$0.147 < |\delta|$（無視できる程度），$0.147 \leq |\delta| < 0.330$（小程度），$0.330 \leq |\delta| < 0.474$（中程度），$0.474 \leq |\delta|$（大程度）



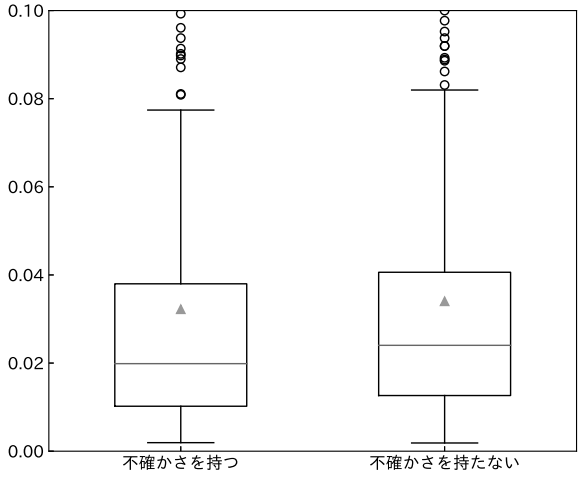
(a) データセット全体



(b) Java



(c) C/C++



(d) Python

図 4.2　ソースコード 1 行あたりに含まれるコードスメルの数の分布

表 4.7　頻出度上位 10 件のコードスメル検出ルールとその頻度（データセット全体，不確かさを持たない）

検出ルール	頻度
関数名は命名規則に従うべきである (C/C++)	1483
メソッド名は命名規則に従うべきである	1365
行は長すぎるべきではない	1091
マジックナンバーを使用すべきではない	698
Docstring を定義すべきである	543
1 行に複数の文を配置するべきではない	522
制御構造には中括弧を使用すべきである	509
行は長すぎるべきではない	506
関数は複雑すぎるべきではない	355
関数名は命名規則に従うべきである (Python)	271

表 4.8　頻出度上位 10 件のコードスメル検出ルールとその頻度（データセット全体，不確かさを持つ）

検出ルール	頻度
定数名は命名規則に従うべきである	2363
行は長すぎるべきではない	2084
マジックナンバーを使用すべきではない	1213
関数名は命名規則に従うべきである	1014
Docstring を定義すべきである	925
制御構造には中括弧を使用すべきである	814
1 行に複数の文を配置するべきではない	384
三項演算子を使用すべきではない	351
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	323
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	315

表 4.9　頻出度上位 10 件のコードスメル検出ルールとその頻度（Java，不確かさを持たない）

検出ルール	頻度
行は長すぎるべきではない	2894
マジックナンバーを使用すべきではない	1997
制御構造には中括弧を使用すべきである	1623
定数名は命名規則に従うべきである	888
三項演算子を使用すべきではない	620
1 行に複数の文を配置するべきではない	431
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	381
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	378
著作権およびライセンスを示すヘッダで始まる必要がある	374
テストによって十分なカバレッジを持つべきである	369

表 4.10　頻出度上位 10 件のコードスメル検出ルールとその頻度（Java，不確かさを持つ）

検出ルール	頻度
マジックナンバーを使用すべきではない	3702
行は長すぎるべきではない	2294
制御構造には中括弧を使用すべきである	1923
1 行に複数の文を配置するべきではない	861
三項演算子を使用すべきではない	812
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	485
コードの一部をコメントアウトするべきではない	457
定数名は命名規則に従うべきである	437
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	436
例外ハンドラは元の例外を維持するべきである	388

表 4.11　頻出度上位 10 件のコードスメル検出ルールとその頻度（C/C++，不確かさを持たない）

検出ルール	頻度
関数名は命名規則に従うべきである	4490
メソッド名は命名規則に従うべきである	2178
関数は複雑すぎるべきではない	1090
1 行に複数の文を配置するべきではない	881
メソッド/関数の認知的複雑性は高すぎるべきではない	721
すべてのソースファイルに著作権およびライセンスヘッダを定義すべきである	382
FIXME タグを対処するべきである	351
テストによって十分なカバレッジを持つべきである	253
ソースファイルは十分なコメント行を持つべきである	115
ファイルは複雑すぎるべきではない	48

表 4.12　頻出度上位 10 件のコードスメル検出ルールとその頻度（C/C++，不確かさを持つ）

検出ルール	頻度
関数名は命名規則に従うべきである	3526
メソッド名は命名規則に従うべき	1480
関数は複雑すぎるべきではない	765
メソッド/関数の認知的複雑性は高すぎるべきではない	592
1 行に複数の文を配置するべきではない	554
すべてのソースファイルに著作権およびライセンスヘッダを定義すべきである	376
テストによって十分なカバレッジを持つべきである	260
ソースファイルは十分なコメント行を持つべきである	112
FIXME タグを対処するべきである	99
ファイルは複雑すぎるべきではない	39

表 4.13　頻出度上位 10 件のコードスメル検出ルールとその頻度（Python，不確かさを持たない）

検出ルール	頻度
Docstring を定義すべきである	3160
行は長すぎるべきではない	2139
関数の認知的複雑性は高すぎるべきではない	1096
文字列リテラルは重複すべきではない	644
関数名は命名規則に従うべきである	452
SystemExit を再度 raise すべきである	429
関数は return 文をあまり多く含むべきではない	423
「print」文は使うべきではない	413
テストによって十分なカバレッジを持つべきである	383
1 行に複数の文を配置するべきではない	367

表 4.14　頻出度上位 10 件のコードスメル検出ルールとその頻度（Python，不確かさを持つ）

検出ルール	頻度
行は長すぎるべきではない	3401
1 行に複数の文を配置するべきではない	2288
Docstring を定義すべきである	2191
関数の認知的複雑性は高すぎるべきではない	677
関数名は命名規則に従うべきである	513
関数やメソッドは空であってはならない	450
「print」文は使うべきではない	387
テストによって十分なカバレッジを持つべきである	380
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	324
文字列リテラルは重複すべきではない	299

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

ソースコード品質への影響として，複雑度，結合度の 2 つの品質について調査する．複雑度が高いことは，ソースコードのテストパターンの数が多いことを示している．そのため，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．結合度が高いことは，多くの他のクラスやメソッドに依存していることを示しているため，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．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 を答えるために，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 5.1 に示す．

手順 i　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 ii　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 iii　品質メトリクスを求める．
手順 iv　マンホイットニーの U 検定で有意差検定する．

手順 i～ 手順 ii は，RQ1 の手順 1，手順 2 と同様であるため，本章では省略する．

手順 iii では，複雑度，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 (注 6) を利用した．手順 ii 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

(注 6)：http://www.techmatrix.co.jp/product/understand/index.html

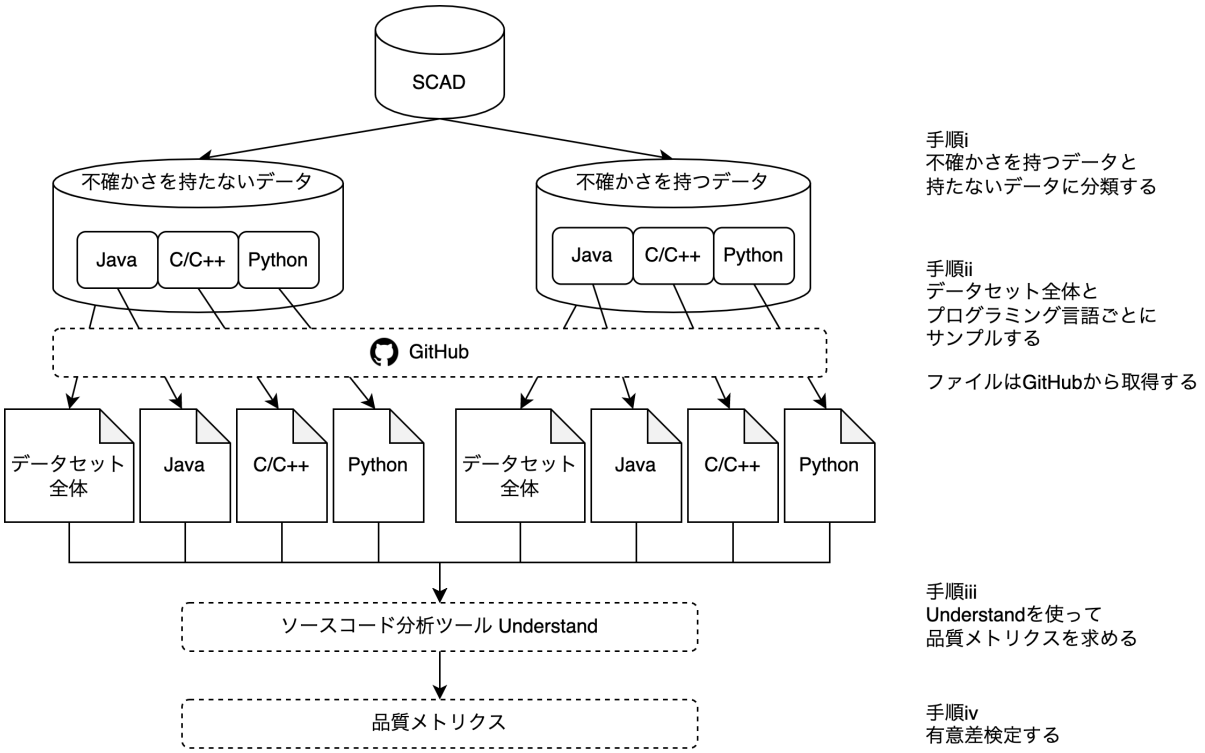


図 5.1　RQ2 における調査方法の概要図

に対して計算される．また，複雑度，結合度のメトリクスは，ファイル行数が多いほど，複雑度，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．

複雑度は，2.4 章で述べた McCabe の循環的複雑度の用いる．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 行あたりの複雑度とする．結合度は，2.4 章で述べた Chidamber らのクラス結合度を用いる．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 行あたりの結合度とする．複雑度，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.

手順 4 では，マン・ホイットニーの U 検定を用いて有意差を比較する．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に有意差をマン・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 [21].

5.3 結果

SCAD 全体やプログラミング言語ごとに，複雑度の統計量を表 5.1，表 5.2 に示す．結合度の統計量を表 5.3，表 5.4 に示す．これらの統計量をもとに作成した複雑度に対する箱ひげ図を図 5.2 に，結合度に対する箱ひげ図を図 5.3 に示す．図の三角マーカーは平均値を示している．また，マン・ホイットニーの U 検定と Cliff の $|\delta|$ の結果を表 5.5，表 5.6 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．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．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 に対する結合度は有意差が出ていない

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，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．

結果として，不確かさをソースコードに持つほど，C/C++，Python に対して複雑度は高く，C++，Python に対して結合度はより高くなることが明らかになった．また，Java に対する結合度では，不確かさを持たないソースコードの方が結合度は高くなる事が明らかになった．

表 5.1　不確かさを持つソースコードに対する 1 行あたりの複雑度の統計量

	データセット全体	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 5.2　不確かさを持たないソースコードに対する 1 行あたりの複雑度の統計量

	データセット全体	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 5.3　不確かさを持つソースコードに対する 1 行あたりの結合度の統計量

	データセット全体	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

表 5.4　不確かさを持たないソースコードに対する 1 行あたりの結合度の統計量

	データセット全体	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 5.5　1 行あたりの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

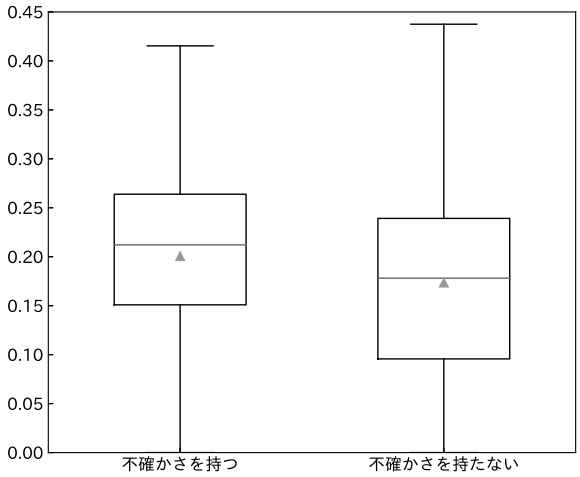
対象データ	有意差	効果量 $\|\delta\|$
データセット全体	✓	0.190
Java		0.041
C/C++	✓	0.140
Python	✓	0.190

効果量の大きさ：$0.147 < |\delta|$（無視できる程度），$0.147 \leq |\delta| < 0.330$（小程度），$0.330 \leq |\delta| < 0.474$（中程度），$0.474 \leq |\delta|$（大程度）

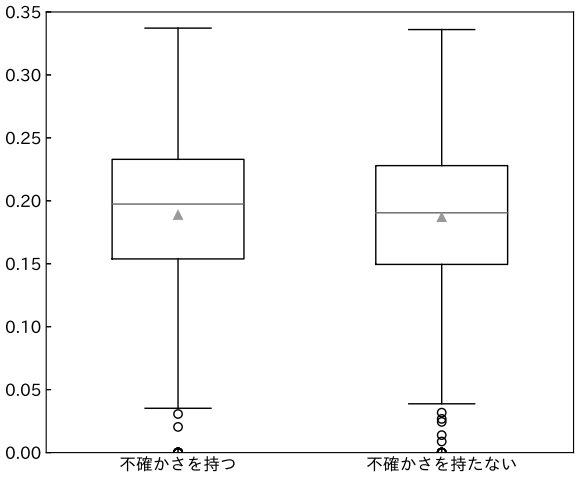
表 5.6　1 行あたりの結合度に対するマン・ホイットニーの U 検定の結果と Cliff の $|\delta|$ による効果量

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

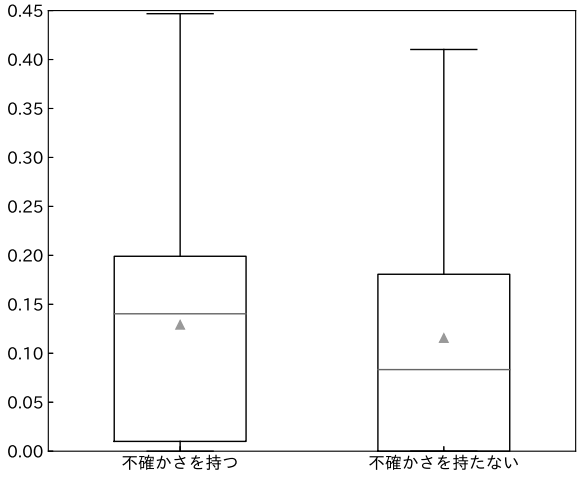
効果量の大きさ：$0.147 < |\delta|$（無視できる程度），$0.147 \leq |\delta| < 0.330$（小程度），$0.330 \leq |\delta| < 0.474$（中程度），$0.474 \leq |\delta|$（大程度）



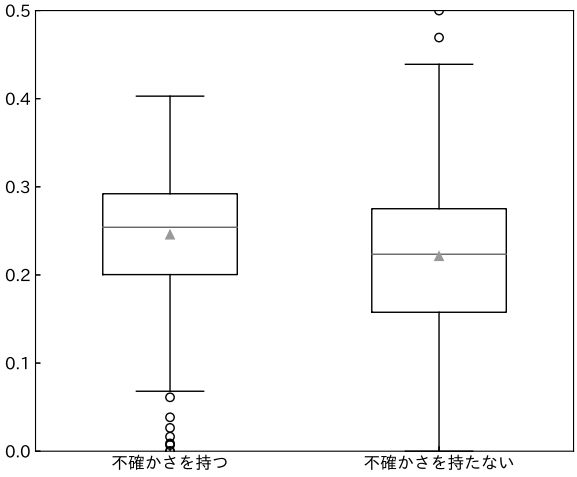
(a) データセット全体



(b) Java

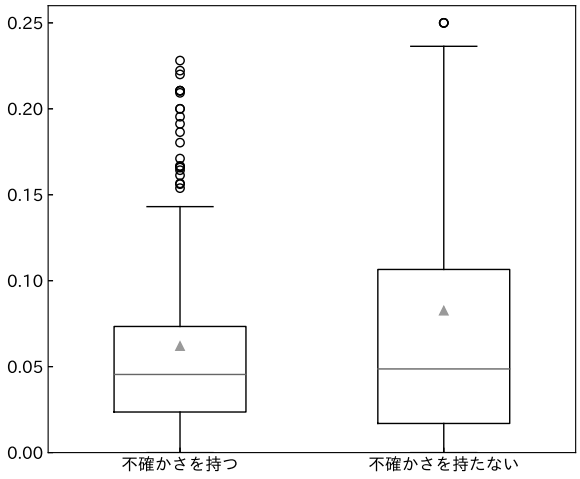


(c) C/C++

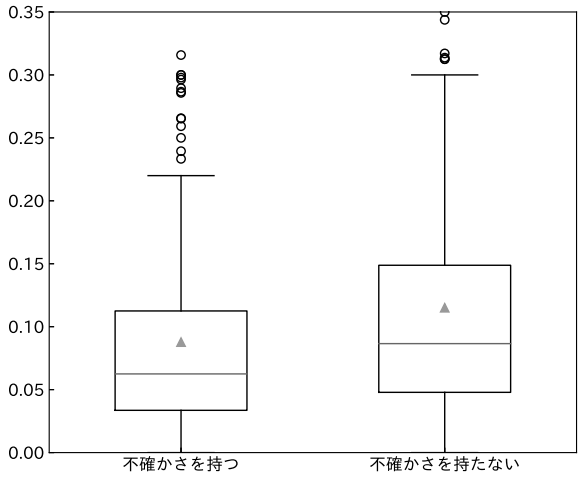


(d) Python

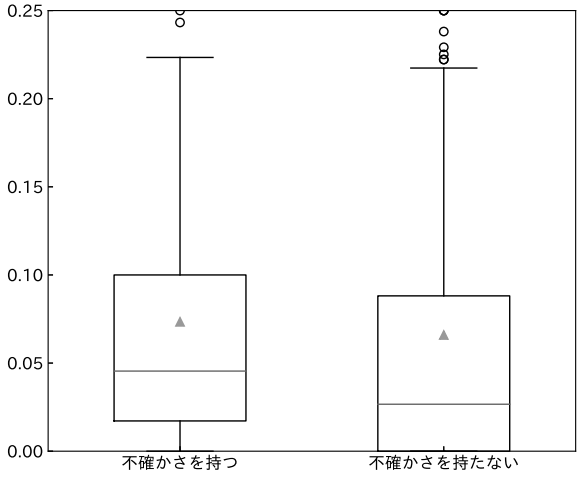
図 5.2　ソースコード 1 行あたりの複雑度の分布



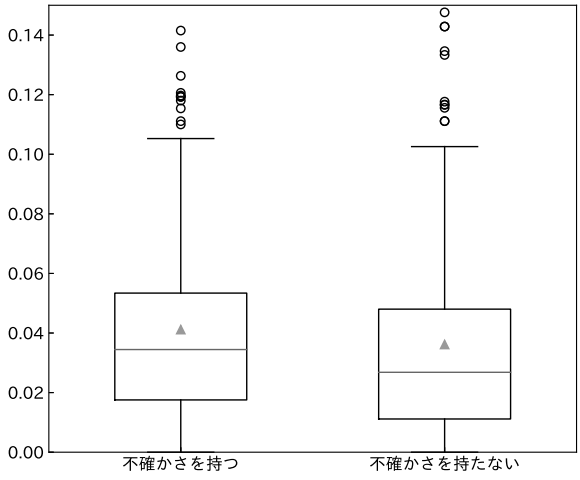
(a) データセット全体



(b) Java



(c) C++



(d) Python

図 5.3　ソースコード 1 行あたりの結合度の分布

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．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．

表 4.10～表 4.14 より，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ていることが分かる．具体的には，Java は 7 件，C/C++は 10 件全て，Python は 8 件一致している．つまり，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．

一方，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には注目しておくべきである．頻出度上位 10 件のうち，片方のみ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

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，もしくは，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 における複雑度，C++，Python における結合度では，ともに有意差が見られ，Java では複雑度，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 の結合度について，不確かさを持たない方が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

7.1 外的妥当性

今回の調査では，GitHub 上でスター数が少なくとも 10 個以上所持している 106304 個のリポジトリから作成されたデータセットを利用している．これらのリポジトリで，全てのソフトウェアにおける特徴を網羅できているかは定かではない．特に，OSS ではなく企業内で開発されているソフトウェアでは異なる可能性がある．

RQ2 に対する調査結果のように，不確かさによる影響はプログライミング言語によって異なる可能性がある．本論文で調査した Java，C 言語，C++，Python の 4 つのプログラミング言語以外に適用できるかどうかは，調査する必要がある．

7.2 内的妥当性

RQ1，RQ2 に対する調査においてコード行数による影響を考慮するために，1 行あたりのメトリクスの値を調査し議論した．また，マンホイットニーの U 検定による有意差検定，Cliff の $|\delta|$ による効果量測定により有効性を数値的に評価した上で議論している．

7.3 構成概念妥当性

本論文で利用したデータセットは，データセットの作成時に取得したリポジトリ内のソースコードから全てのソースコードコメントが抽出されていない可能性がある．このデータセットは Doxygen で生成されているため，生成結果は Doxygen の処理に依存している．また広く利用されているデータセットではないことから，妥当性への脅威となりえる．

不確かさの識別には，先行研究で報告されている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た．これらのキーワードと特徴単語を，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．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．

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

本研究でソースコードコメントに着目したように，開発者がソースコードコメントに記載する内容として注目されている対象として，Self Admitted Technical Debt(SATD) がある．SATD とは，開発者が意図的に導入した技術的負債のことであり，コードコメントに明示的に記載されているものを指す [22]．SATD も不確かさも，開発者が意図的にソースコードコメントに内容を記載するという点が一致している．そのため，SATD に対する調査結果や，調査方法としてアプローチが参考になる可能性がある．

Potdar と Shihab は，ソースコードコメントのうち SATD である可能性が高いパターンを調査した [22]．結果として，hack，fixme，workaround for bug を含む 62 個のパターンを発見し，このパターンを用いてソースコードコメントをマイニングすることで SATD を検出できることを示した．

Bavota と Russo は，SATD に関するより大規模な実証研究を行っている [18]．彼らは，検出した SATD をより詳細にカテゴリ分類し，SATD を導入する開発者，取り除く開発者を特定した．また，ソースコードの品質と SATD の数に対する相関関係について調査している．品質メトリクスとして，複雑性，結合性，可読性に注目し，検出された SATD の数とソースコードの品質の低さの間には相関関係がないとしている．

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．つまり，不確かさは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．また，不確かさがあることで，C/C++，Python では複雑度が有意に高く，C++，Python では結合度が有意に高くなった．これは，不確かさが，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．一方，Java では不確かさを持たない方が，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．その理由として，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．また，不確かさによる影響はプログラミング言語によって異なることを示唆している．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系水野修教授ならびに崔恩瀞助教に厚く御礼申し上げます．また，本報告書執筆にあたり研究生活を共にし，貴重な助言を多数頂いたソフトウェア工学研究室の皆さん，いつも研究室で温かく迎えてくれたあくあたん，学生生活を通じて著者の支えとなった家族や友人に深く感謝致します.

参考文献

[1] 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,” 研究報告ソフトウェア工学（SE），vol.2017-SE-196, no.29，pp.1–8，2017.

[2] 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.

[3] 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.

[4] N. Ubayashi, Y. Kamei, and R. Sato, “When and Why Do Software Developers Face Uncertainty?,” Proceedings of the 19th International Conference on Software Quality, Reliability and Security (QRS), pp.288–299, 2019.

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

[6] W. Cunningham, “The WyCash portfolio management system,” Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pp.29–30, 1992.

[7] 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.

[8] V. Lenarduzzi, A. Sillitti, and D. Taibi, “A Survey on Code Analysis Tools for Software Maintenance Prediction,” Proceedings of the International Conference in Software Engineering for Defence Applications (ICSE), pp.165–175, 2018.

[9] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, “DECOR: A Method for the Specification and Detection of Code and Design Smells,” IEEE Transactions on Software Engineering, vol.36, no.1, pp.20–36, 2010.

[10] J. Pereira dos Reis, F. Brito e Abreu, G. de Figueiredo Carneiro, and C. Anslow, "Code Smells Detection and Visualization: A Systematic Literature Review," Archives of Computational Methods in Engineering, vol.29, no.1, pp.47–94, 2022.

[11] D. Garlan, "Software engineering in an uncertain world," Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER), p.125–128, 2010.

[12] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," Proceedings of the 34th International Conference on Software Engineering (ICSE), pp.573–583, 2012.

[13] 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.

[14] N. Esfahani, K. Razavi, and S. Malek, "Dealing with uncertainty in early software architecture," Proceedings of the 20th International Symposium on the Foundations of Software Engineering (FSE), pp.1–4, 2012.

[15] 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.

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

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

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

[19] B. Gelman, B. Obayomi, J. Moore, and D. Slater, "Source Code Analysis Dataset," Data in Brief, vol.27, p.104712, 2019.

[20] T. Sonnekalb, T. Heinze, and P. Mäder, "Deep security analysis of program code: A systematic literature review," Empirical Software Engineering, vol.27, pp.1–39, 2022.

[21] 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.

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