卒 業 研 究 報 告 書

題　目　　学生が書いたプログラムの視認性向上に関する調査

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　　19122029

氏　　名　　辰岡 那由太

令和5年2月13日提出

学生が書いたプログラムの視認性向上に関する調査

令和 5 年 2 月 13 日　　19122029　辰岡 那由太

概　要

プログラムの視認性とは，プログラムの可読性を示す 1 つの指標であり，プログラムの瞬間的な認識，つまり一目見た時の見やすさを表す度合いを指す．学生が書いたプログラムの品質と最終的なプログラムの品質には強い正の相関関係があること [1] や，プログラムの可読性と品質にも関係があることが分かっているため [2]，学生のプログラムの視認性を改善することで，プログラムの品質の向上が期待できる．しかし学生の場合，できるだけ早く問題を解決しようとすることが多いため [3]，視認性まで意識できている学生は少ないと考えられる．そのため，プログラムの視認性に関係のあるメトリクスを明らかにすることで，学生がそのメトリクスに対して注意することで，プログラムの視認性の向上が見込まれる．さらに視認性が向上することで最終的なプログラムの品質の向上まで期待できる．

そこで，本研究では，プログラムの視認性の向上に貢献できるメトリクスについて，既存のフォーマッターを用いて調査する．本調査では，フォーマット後のプログラムを視認性の良いプログラムとし，フォーマッターによって訂正された行数の割合が高いプログラムほど，視認性の良くないプログラムだと仮定する．この仮定に基づいて分類した視認性の良いファイル群と視認性の悪いファイル群に対して，フォーマットルールを適切に変更したフォーマットを行うことで，目的のメトリクスを検出する．

調査の結果，インデントの不揃いの整頓，演算子の前後などの空白の整頓，ブロック文の改行のメトリクスに対して，視認性の良いファイル群と良くないファイル群の間で有意差が確認できた．特にインデントの不揃いの整頓の項目に関しては，2 つのファイル群の間で大きな違いが見受けられた．この結果から，特にインデントの不揃いに対して意識がけすることで，プログラムの視認性を改善できることが明らかになった．

目　次

1. 緒言

プログラムの視認性とは，プログラムの可読性を示す 1 つの指標であり，プログラムの瞬間的な認識，つまり一目見た時の見やすさを表す度合いを指す．学生が書いたプログラムの可読性とプログラムの品質は強い正の相関関係がある [1] ため，学生のプログラムの視認性を向上させることで，プログラムの品質の向上が期待できる．さらに，近年，ソフトウェアの開発において，より多くのコストが保守に注ぎ込まれるようになった．その結果，ソフトウェアの保守は，ソフトウェア開発ライフサイクルコストの大きな部分を占めており [4]，プログラムを新しく書くコストより，既存のプログラムを読んで理解するコストが高くなりつつある．そのため，フォーマッターといった静的解析ツールに限らず，動的解析を用いて視認性，可読性を向上させる研究が行われてきた [5][6][7][8].

しかし学生の場合，できるだけ早く問題を解決しようとすることが多いため [3]，視認性まで意識できている学生は少ないと考えられる．学生のプログラムの，良くない視認性を向上させることは，プログラムを読んで理解するコストの削減や，プログラムの品質の向上で，大きな意味を持つと考えられる．

そこで，本研究では，学生が書いたプログラムの視認性に注目し，プログラムの視認性の向上に貢献できるメトリクスについて，フォーマッターを用いて調査する．視認性の評価方法は人間の主観評価が多く，個人差による結果の不安定性も大きい [9] ため，本調査ではフォーマッターによってフォーマットされた後のプログラムを「視認性の良いプログラム」とし，フォーマッターによって訂正された行数の割合が高いプログラムほど，視認性の良くないプログラムだと仮定する．この仮定に基づいて，分類した視認性の良いファイル群と視認性の悪いファイル群に対して，フォーマットルールを適切に変更したフォーマットを行うことで，目的のメトリクスを検出する．

調査の結果，インデントの不揃いの整頓，演算子の前後などの空白の整頓，ブロック文の改行のメトリクスに対して，統計的な有意差が確認できた．特にインデントの不揃いの整頓の項目に関しては，2 つのファイル群の間で大きな違いが見受けられた．この結果から，特にインデントの不揃いに対して意識がけすることで，プログラムの視認性を改善できることが明らかになった．

以降では本論文の章構成を示す．まず，第 2 章では本論文の調査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，調査データ，使用するフォーマッター，フォーマッターのルール設定，調査方法並びに調査結果について述べる．第 4 章では，視認性向上のための支援と妥当性への脅威を説明する．第 5 章では関連研究，最後に第 6 章で結言を述べる．

2. 研究背景

2.1 プログラムの視認性

プログラムの視認性とは，プログラムの可読性 [10]，プログラムを人間が読んだ時の，その目的や処理の流れの理解しやすさそ示す１つの指標であり，プログラムの瞬間的な認識，つまり一目た時の見やすさを表す度合いを示す.

ソフトウェアは，知的財産を形成しつつあり，プログラムを読む比重はますます大きくなりつつある．近年，ソフトウェアの開発において，より多くのコストがソフトウェアの保守に注ぎ込まれるようになった．その結果，ソフトウェアの保守は，ソフトウェア開発ライフサイクルコストの大きな部分を占めている．プログラムを新しく書くコストより，既存のプログラムを読んで理解するコストの方が高くなりつつあるため，プログラムの可読性は，ソフトウェア開発にとって，重要なファクターである [4].

可読性に作用するものとして,「何を意味しているのか判断しやすい変数名，関数名をつける」,「ネストを減らす」などが取り上げられる [11]．学生の場合，プログラムの品質を認識せず，できるだけ早く問題を解決しようとすることが多い [3] ため，これらの項目は意識しない.

視認性はプログラムを一目見た時の瞬間的な認識であるため，プログラムの意味や流れを理解する必要はなく，静的解析を用いて定量的に判別することができる．しかし視認性は各々の感性によるところが大きいため，一概に視認性の良し悪しの判断は困難である.

2.2 フォーマッター

フォーマッターは別名コーディング規約チェックツールで，静的解析ツールの１つである．フォーマッターは開発者がプログラミング言語で記述したプログラムを入力として受け取り，一定のルールに従って整形し，整形されたプログラムを出力するツールであり，括弧やスペースの使い方などのスタイルを統一するような機能を持つツールである．静的解析とはプログラムを動作させることなく解析する手法で，それに対し実際にプログラムを動かして解析する手法を動的解析という [12]．動的

解析は実際に発生する問題や不具合の検出に適しており，静的解析は不具合を引き起こすコード上の欠陥箇所や命名規則違反に書式の検証などに役立つとされており，互いに補完的な役割を担う [13].

Java プログラムのフォーマッターには checkstyle (注 1)，HTML を対象とするものには Markup Validation Service (注 2) など存在するが，C 言語を対象としたフォーマッターは，我々が知る限り，多く存在しない.

C 言語を対応とし，ユーザが各自ででフォーマットルールをカスタマイズできるフォーマッターとして CX-checker (注 3) がある．しかし，このフォーマッターは大学で作成された研究用のプロダクト [14] で，インストールページが更新されておらず，一部利用方法ページがアクセス不可とされているため，個人での利用が厳しいと判断した.

Artistic Style (注 4) は C/C++/Objective-C/C #/Java で利用可能なフォーマッターである．このフォーマッターは，CX-checker よりも頻繁にアップデートされ，利用方法の説明も充実しており，ユーザが各自でフォーマットルールを自由に設定できるオプションも存在する．Artistic Style は入力としてプログラムを受け取り，ユーザが設定したフォーマットルールに従ってフォーマットされたプログラムを，入力ファイルに上書き保存する形で出力する．この際，フォーマット前のファイル名を書き換えて新規保存する仕様がある.

clang-format (注 5)は，LLVM コンパイラの一つである clang (注 6) に付属している，C/C++/Objective-C のプログラムを自動的にフォーマットするツールでである．このツールも Artistic Style と同じく頻繁にアップデートされ，利用方法も細かく記述されている．clang-format は LLVM，Mozilla などのコーディング規約に準拠した上で，'.clang-format' という名前の設定ファイルに yaml 形式で設定を記述することで，ユーザが各自でインデント幅，空白の挿入，波括弧の制御方法など，自分好みのフォーマットルールを細かく設定することができる．clang-format は入力としてプログラムを渡すと，ユーザが設定したフォーマットルールに従ってフォーマットされた

(注 1)：https://checkstyle.org
(注 2)：https://validator.w3.org
(注 3)：https://kariyasiesta.github.io/kariyasiesta-manual/
(注 4)：https://astyle.sourceforge.net
(注 5)：https://clang.llvm.org/docs/ClangFormat.html
(注 6)：https://clang.llvm.org

ファイルを，標準出力で表示する．

3. 調査

3.1 調査目的

本研究の目的は，学生が書いたプログラムに対して， 2.1 節で説明したプログラムの視認性を改善させるメトリクスを明らかにすることである．

学生のプログラムの品質と最終プログラムの品質には強い正の相関関係があり [1]，また，プログラムの可読性と品質にも関係がある [2] ため，学生のプログラムの視認性を改善することで，プログラムの品質の向上が期待できる．

プログラムの視認性に関する評価基準が一般に公開されることは少なく，また公開されていたとしても人間の主観評価が多い [9]．しかし主観的な評価方法では，個人差による結果の不安定性が大きく，フィードバックに手間や時間がかかる [9].

そこで，本研究では，フォーマッターによってフォーマットされた後のプログラムを「視認性の良いプログラム」と仮定し，フォーマッターを用いて調査対象のプログラムを様々なフォーマットルールの下でフォーマットを行い，どのメトリクスが視認性に影響を与えるかについて調査する．

その結果，プログラムの視認性に関係のあるメトリクスを明らかにすることで，学生がそのメトリクスに対して注意し，視認性の向上が見込まれ，最終プログラムの品質の向上まで期待できる．

3.2 調査対象データセット

本学の学部 2 年生前学期で行われる授業，ソフトウェア演習 2 の 2008 年〜2021 年の 13 年分のデータに対して調査を行う．データセットには 1 年ごとに 80 人前後の学生と 13 個の課題が存在し，合計で 12580 個のファイルがある．ソフトウェア演習 2 は基本的なデータ構造やアルゴリズムについて C 言語を用いて行う演習であり，2 年生の演習科目ということでプログラミングに慣れていない学生のプログラムが提出されている．

3.3 調査で利用するフォーマッター clang-format

視認性に関与するメトリクスを検出するにあたって，数多くのデータに対して複数回，違うフォーマットルールでのフォーマットを行う必要がある．したがって，2.2 節で説明した Artistic Style は，出力ファイルが入力ファイルに上書きされる仕様があるため，本調査では適してないと判断した．

調査で利用するフォーマッターとして，2.2 節で説明した clang-format を選択した．clang-format はフォーマット済みの出力ファイルが標準出力に表示されるため，Artistic Style での問題をクリアしている．さらにインデント幅，空白の挿入，波括弧の制御方法などを個別に設定できるため，本研究に用いるのに適切だと判断した．

本研究では標準出力に表示されたフォーマット済みのプログラムを，直接次の処理の入力と設定することで，標準出力に表示されたプログラムを新たに保存することなくフォーマット前後のプログラムの比較を行う．

3.4 フォーマッターのルール設定

clang-format では'.clang-format' という名前の設定ファイルに yaml 形式で設定を記述することで，プログラムのフォーマット，コーディングスタイルの内の見た目に関するものを，ユーザが各自で設定したルールに従って細かく整形することが可能であり，その中で本研究目的に合う適切なルール設定を行うことが重要になる．本研究では，視覚的なメトリクスにのみフォーマットを行いたいので，ルール設定もそれを満たすように設定する．またプログラミングには，プログラムの挙動には関係はないが，プログラムを記述する際に表れる自由度，いわゆる自分ルールが存在する．企業などではこれらの自由度によって，他者がコードを読みづらくなることを防ぐためにコーディング規約を社内で適用している場合が多い．フォーマッターのルールを設定する際には，ある程度の自由度を持たせた上で，視認性に影響を与えるメトリクスについて調査が行えるように注意が必要である．実際に設定した clang-format のフォーマットルールを表 3.1 に示す．

長い 1 行を改行する際に，どこで改行を挟むかは，個人の裁量，自分ルールが表れると予想し，1 行の最大文字数を無制限に設定することで，フォーマッターによる改行が行われないようにした．またインデント幅に関しては，本学で学生に対して

推奨されている統合開発環境である Eclipse[注 7] でデフォルトの値として設定されている半角スペース 4 で設定した．また，ソースコードエディタの Visual Studio Code[注 8] においてもインデントのデフォルト値は半角スペース 4 である．ここでインデント幅を変更することによる視認性の阻害を検出するために，IndentCaseLabel 項目でインデントの量を 1 レベルとしている．プログラムを記述する際の自由度を許容するために．Align 項目を全て整列をせず，波括弧の改行スタイルを統一し，制御構文の前後で改行を行わないよう設定した．

(注 7)：https://www.eclipse.org
(注 8)：https://code.visualstudio.com

表 3.1: clang-format のフォーマットルール設定

項目	設定	内容
AlignArrayOfStructures	none	構造体の配列の初期化の際にフィールドを整列させない.
AlignConsecutiveAssignments	None	複数行連続している代入演算子を整列させない.
AlignConsecutiveBitFields	None	連続したビットフィールドのコロンを整列させない.
AlignConsecutiveDeclarations	None	連続した宣言を整列させない.
AlignConsecutiveMacros	Consecutive	連続したマクロ定義を整列させる.
AlignEscapedNewlines	DontAlign	改行エスケープを整列させない.
AlignTrailingComments	false	複数行連続している末尾のコメントを整列させない.
AlwaysBreakBeforeMultilineStrings	false	複数行の文字列リテラルの前で改行しない.
ColumnLimit	0	1 行の最大文字数は制限なし.
ConstructorInitializerIndentWidth	4	コンストラクタ初期化を改行した場合のインデントの幅を半角スペース 4 とする.
DeriveLineEnding	true	改行コードをプログラムから自動的に判断する.
IndentCaseBlocks	true	case ラベルのブロックを case ラベルから 1 レベルインデントする.
IndentCaseLabels	true	case ラベルを switch 文から 1 レベルインデントする.

表は次ページに続く

前ページからの続き

項目	設定	内容
IndentWidth	4	インデント幅を半角スペース 4 とする (eclipse のデフォルト値．vscode も同様).
IndentWrappedFunctionNames	true	関数の宣言や定義の型の後で改行されている場合，インデントする.
Language	Cpp	フォーマット対象の言語は C,C++.
MaxEmptyLinesToKeep	1	連続した空行を 1 行まで保持する.
SpaceAfterCStyleCast	true	C スタイルのキャストの後に空白を入れる.
SpaceBeforeAssignmentOperators	true	代入演算子の前に空白を入れる.
SpaceBeforeCpp11BracedList	true	初期化子リストの前に空白を入れる.
Standard	Auto	C++のバージョンをコードに基づいて自動で判断する.
TabWidth	4	tab によるインデントの場合は半角スペース 4 とする.
AfterCaseLabel	false	case ラベルの後の波括弧で改行しない.
AfterControlStatement	Never	if，for，while，switch などの制御構文の波括弧で改行しない.
AfterFunction	false	関数定義の名前の後の波括弧で改行しない.
AfterNamespace	false	名前空間の後の波括弧で改行しない.
AfterStruct	false	構造体の名前の後の波括弧で改行しない.
AfterExternBlock	false	extern ブロックの名前の後の波括弧で改行しない.

表は次ページに続く

前ページからの続き

項目	設定	内容
BeforeCatch	false	catch の前で改行しない.
BeforeElse	false	else の前で改行しない.
BeforeWhile	false	while の前で改行しない.
IndentBraces	false	改行した波括弧をインデントしない.

表終了

3.5 調査方法

以下の手順でプログラムの視認性の改善に貢献できるメトリクスを以下の手順で調査する．この手順を図 3.1 に示す.

手順 1　3.2 節で説明したデータセットに対してフォーマットを行う.
手順 2 フォーマット前の行数に対するフォーマッターによる訂正が行われた行数の割合を計算する.
手順 3 訂正行数割合を元に「視認性の良いファイル群」と「視認性の良くないファイル群」に分類する.
手順 4 フォーマッターと diff コマンドを用いてメトリクスを検出する.
手順 5 マン・ホイットニーの U 検定で有意差検定を行う.

手順 1 では，3.2 節で述べた全てのデータに対して clang-format によるフォーマットを行う．フォーマットの際のルールには 3.4 節で述べたルールを適用する．このフォーマットルールはプログラムを記述する際の自由度を許容した上で，視覚的なメトリクスに対してフォーマットを行うように設定する.

手順 2 では，フォーマット前の行数に対するフォーマッターによる訂正が行われた行数の割合（以降，訂正行数割合）を計算する．一般的に，フォーマッターによって訂正される行数は，プログラムの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，訂正が行われた行の総数ではなく，1 つのプログラムにおける訂正行数割合を求め比較することにする.

手順 3 では，手順 2 で求めた訂正行数割合を元に,「視認性の良いファイル群」と「視認性の良くないファイル群」の 2 つに分類する．ここで，訂正行数割合が低いプログラムほど視認性の良いプログラムとなる．全てのファイルの訂正行数割合の第 1 四分位数以下のファイルを「視認性の良いファイル群」，第 3 四分位数以上のファイルを「視認性の良くないファイル群」と位置付ける.

手順 4 では，視認性の良いファイル群と，視認性の良くないファイル群から，フォーマッターを用いて視認性に影響を与えると思しきメトリクスを調査する．調査では 3.4 節で説明したフォーマットルールから，適切にルールを変更し，改めてフォーマッ

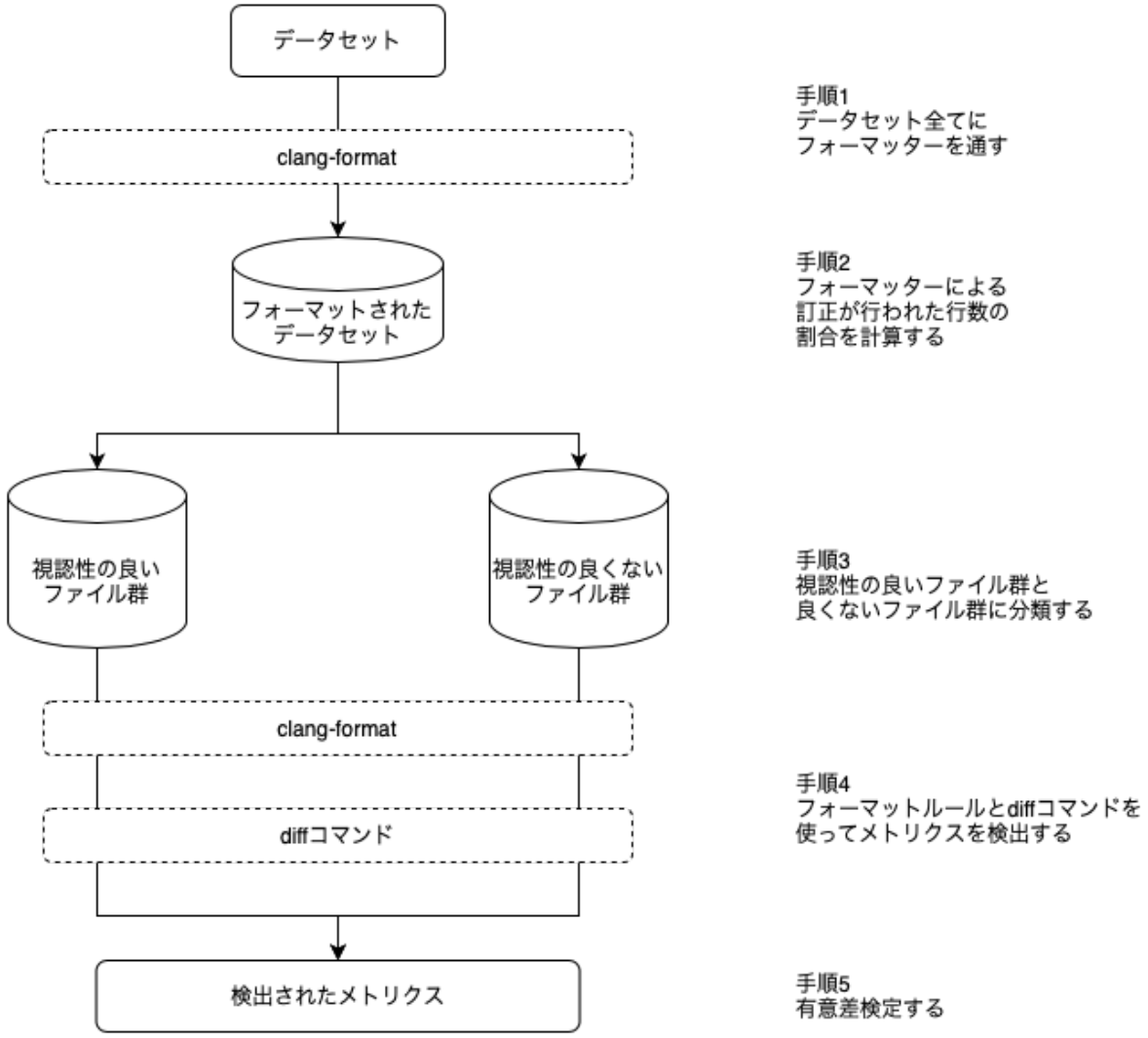

図 3.1　調査方法の概要図

トを行い，調査するメトリクスごとに訂正行数割合を求める．しかし clang-format はその仕様上，必ず LLVM などのコーディング規約スタイルに準拠する必要があり，特定のメトリクスに対して全くフォーマットを行わないという操作が不可能である．そこで，本研究では視認性に影響を与えると思しきメトリクスを検出するために diff コマンドも用いる．diff コマンドは 2 つのテキストファイルを比較し，異なる箇所を出力するコマンドで，オプションにより空白を無視するなどの限定的な比較が可能である．これらを適切に使用し，視認性に影響を与えると思しきメトリクスを検出する．

手順 5 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．手順 4 で検出した各メトリクスに対して，視認性の良いファイル群と，視認性の良くないファイル群との間で，訂正行数割合に有意差をマン・ホイットニーの U 検定を行う．有意水準を $p = 0.05$ とする．

3.6 結果

手順 1 で全てのプログラムに対してフォーマットを行った．その結果を元に作成した箱ひげ図を図 3.2 に示す．この図の三角マーカーは平均値を示している．第 1 四分位数は 0.655737705 で，第 3 四分位数は 0.776119403 であった．この値を境界として視認性の良いファイル群と良くないファイル群に仕分けた．その結果，視認性の良いファイルの数は 3148，視認性の良くないファイルの数は 3143 であった．各ファイル群に対して，フォーマットルールや diff コマンドのオプションを調整してメトリクスを検出し，各メトリクスごとに訂正行数割合を求めた．これを元に作成した箱ひげ図を，インデントの不揃いを整頓した訂正行数割合を図 3.3 ，演算子の前後などの空白を整頓した訂正行数割合を図 3.4 ，改行が行われた訂正行数割合を図 3.5 に示す．なお，同じ行内でも複数のフォーマットルール違反が見られる場合もあるため，これらの訂正行数割合を合計しても 100%になるとは限らない．また，マン・ホイットニーの U 検定の結果を表 3.2 に示す．チェックマークは有意差（$p < 0.05$）を示している．検定結果より，検出した 3 つのメトリクスに対して有意差が見られた．また，視認性の良くないファイル群からサンプルを取り，動作させたところ，サンプル全てにおいて仕様を満たす正常な動作が確認できた．これは学生がプログラム

の視認性などを考慮せず，問題の解決を優先すると述べる一連の研究 [3]，[15] の結果を支持するものである．

以上の結果から，以下の 3 つのメトリクスが視認性の向上に貢献できることがわかった．

- インデントの不揃いの整頓
- 演算子の前後などの空白の整頓
- ブロック文での改行

つまり，プログラムの視認性を向上させるには，これらのメトリクスについて学生が意識がけすれば良いと言える．3 つのメトリクスの中で，インデントの不揃いを整頓した訂正行数割合は，特に大きな差が見られた．従って，インデントを適切な場所に適切な数挿入することが，最も大きく視認性を向上させるアプローチだと考えられる．

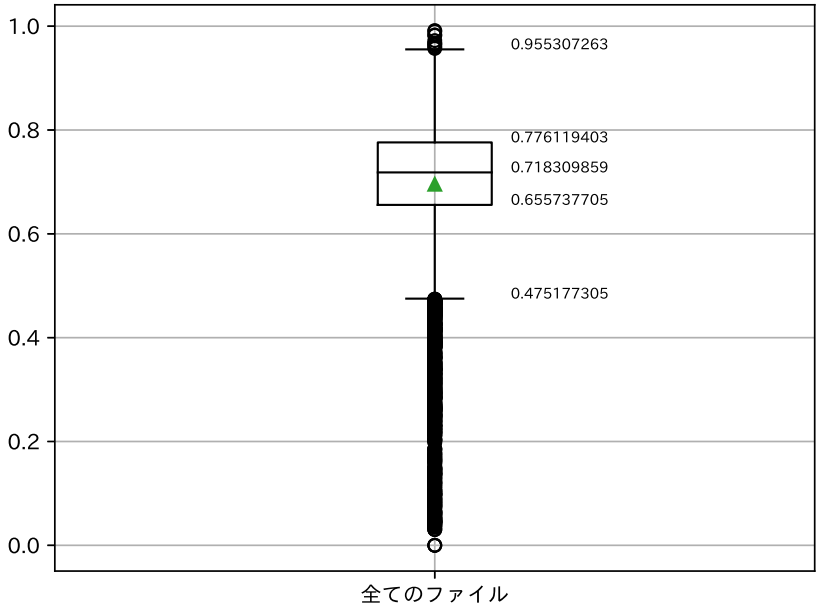

図 3.2　全てのプログラムの訂正行数割合の分布

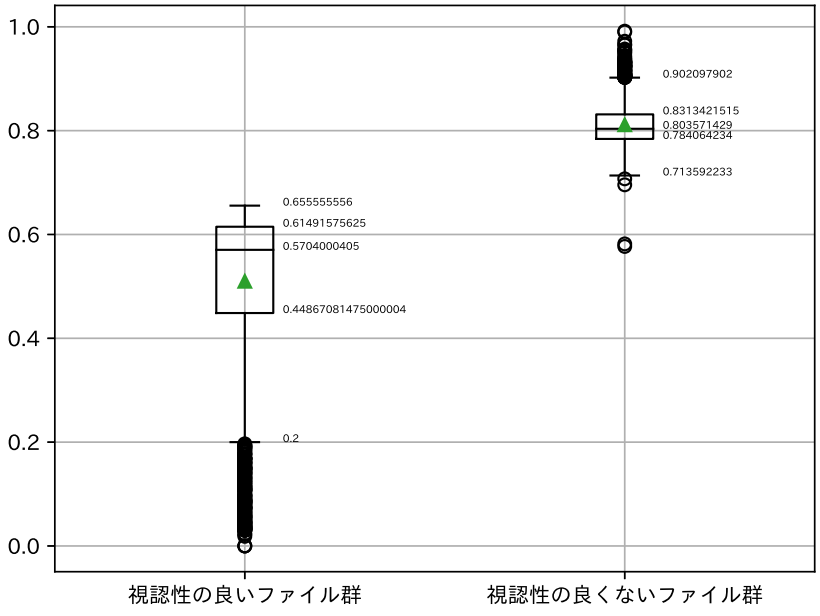

図 3.3　インデントの不揃いの整頓した訂正行数割合の分布

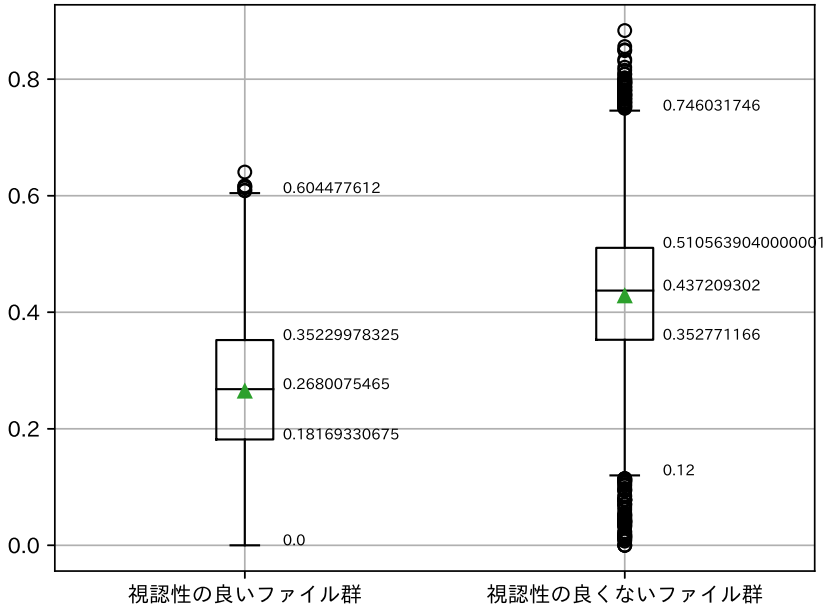

図 3.4　演算子の前後などの空白を整頓した訂正行数割合の分布

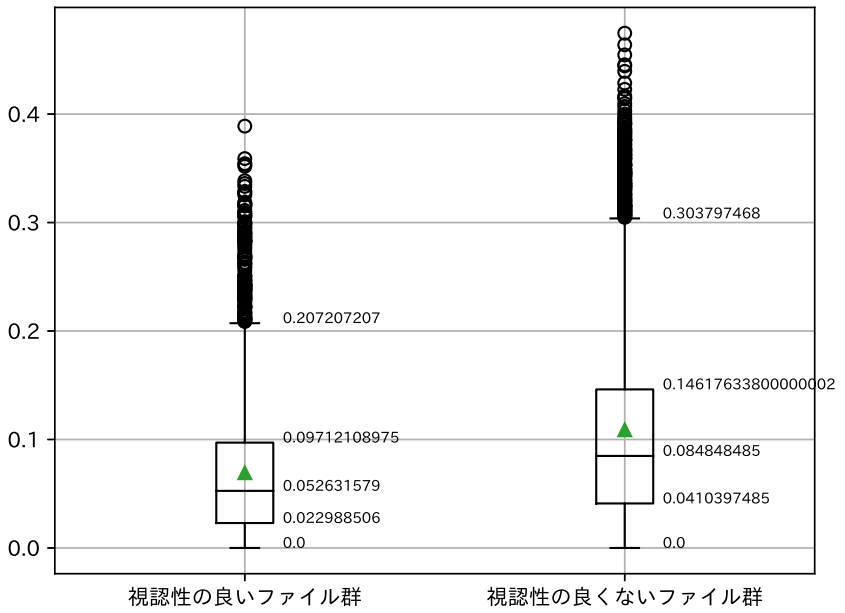

図 3.5　ブロック文での改行が行われた訂正行数割合の分布

表 3.2　訂正行数割合に対するマン・ホイットニーの U 検定の結果

メトリクス	有意差
インデントの整頓	✓
空白の整頓	✓
ブロック文での改行	✓

4. 考察

4.1 視認性を向上させるために必要な支援

本節では，調査結果に基づいて学生が書いたプログラムの視認性を向上させるための支援について考察する．本研究で学生が書いたプログラムの視認性向上に貢献できるメトリクスを調査した結果，インデントの整頓，空白の整頓，ブロック文での改行といったメトリクスが検出された．これらのメトリクスを元に，学生向けのプログラムの視認性を改善するツールへの応用が見込める．ツールの入力として，記述したプログラムを与え，本研究で明らかとなったメトリクスに対してフォーマットを行い，出力として，各メトリクスごとの訂正行数を与える．学生はツールの出力内容を確認することで，特にどのメトリクスによって自分のプログラムの視認性が優れないのかを知ることができ，視認性向上に貢献できるメトリクスへの意識がけが容易になる．

さらに，訂正行数割合を用いて視認性を点数化させることができる場合，プログラムの視認性判定ツールの作成も見込める．このツールは，出力として視認性の度合いが具体的な数値で与えられるため，ツールを利用する学生にとって一目で，自分のプログラムがどれほど視認性に優れているのかを理解しやすいと考えられる．さらに授業内で視認性判定ツールを利用させると，それまでよりも視認性に優れたプログラムが提出されることで，課題の採点者への負担の減少も期待できる．

4.2 妥当性への脅威

4.2.1 外的妥当性

本調査は本学の１つの授業で提出された課題に限定して調査したため，他の授業を対象にした場合，結果が違う可能性がある．従って全ての，学生が書いたプログラムが本研究で明らかになったメトリクスを問題に持つかは定かではなく，これは外的妥当性への脅威である．本研究では，結果が使用ツールに依存しているため，他のツールを使うと違う結果が得られる可能性がある．さらに，検出したメトリクスはプログラミング言語によって異なる可能性があり，本論文で調査したC言語以

外にも当てはまるかどうかは，調査する必要がある．

4.2.2 内的妥当性

本調査において，プログラムの行数による影響を考慮するために，1 行あたりの訂正行数割合を計算し議論した．また，マン・ホイットニーの U 検定による優位さ検定により，有効性を数値的に評価した上で議論している．

4.2.3 内容的妥当性

本調査は，対象のデータセット全てに対し一度フォーマットを行った．しかしメトリクスの検出に用いたのは第 1 四分位数以下と第 3 四分位数以上のデータのみである．これはデータセット全体に対して調査を行えていないと判断でき，内容的妥当性への脅威である．

5. 関連研究

本研究では静的解析ツールであるフォーマッターを用いて視認性の調査を行ったが，異なる手法を用いて可読性についての研究が行われている．

Sarah らは java プロジェクトの可読性についての調査 [16] を静的解析を用いて行った．これは可読性の向上が目的だと明示されたコミットメッセージを分析する手法で行われた．その結果，可読性とは関係のないコミットによってもたらされた問題を修正する傾向があり，その問題は空白や中括弧の訂正などであった．

Butler と Wermelinger は，同じく java プログラムの可読性と品質について，識別子に注目し調査した [17]．その結果，識別子に欠陥がある場合に，プログラムの品質が低いことを特定した．これは静的解析ツールを用いて可読性と品質の関係を明らかにした河崎の研究 [2] と同様の結果を示している．

しかし，これらの研究は，調査対象とする言語，開発者の熟練度，注目するメトリクスなどの点で本研究と異なる．

6. 結言

本研究では，学生が書いたプログラムの視認性に着目し，フォーマッター clang-format を用いて視認性の向上に貢献できるメトリクスを調査した．本学の学部 2 年生が書いた C 言語プログラムを対象に，フォーマッターによってフォーマットされた後のプログラムを「視認性の良いプログラム」と仮定し，clang-format のフォーマットルール設定と diff コマンドを用いてメトリクスを検出し，各メトリクスの調査を行なった．

調査の結果，インデントの不揃いの整頓，演算子の前後などの空白の整頓，ブロック文での改行のメトリクスで，視認性の良いファイル群と視認性の良くないファイル群との間に統計的に有意差があることがわかった．中でも，インデントの不揃いを整頓した訂正行数割合で特に大きな差が確認できた．これはインデントを適切な場所に適切な数挿入することが，最も大きく視認性を向上させるアプローチであることを示唆している．これらのメトリクスについて学生がプログラムを書く際に意識がけすることで，視認性の良いプログラムを書くことができ，結果としてプログラムの品質も向上することが期待できる．

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．本報告書執筆にあたり貴重な助言を多数頂きました，渡邉先輩，ソフトウェア工学研究室の皆さん，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] T. McGill and S. Volet, “An investigation of the relationship between student algorithm quality and program quality,” SIGCSE Bull., vol.27, no.2, pp.44–48, 1995.

[2] 河崎文雄，“静的解析ツールを用いた品質向上施策の検討,” 第 68 回全国大会講演論文集，vol.2006，no.1，pp.165–166，2006.

[3] A. Vizcaíno, J. Contreras, J. Favela, and M. Prieto, “An adaptive, collaborative environment to develop good habits in programming,” Proceedings of the 5th International Conference on Intelligent Tutoring Systems, pp.262–271, 2000.

[4] 中川正樹，早川　栄，玉木裕一，曽谷俊男，“日本語プログラミングの実践とその効果,” 情報処理学会論文誌，vol.35，no.10，pp.2170–2179，oct 1994.

[5] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, “Saner: Composing static and dynamic analysis to validate sanitization in web applications,” Proceedings of 2008 IEEE Symposium on Security and Privacy, pp.387–401, 2008.

[6] T. Ball, “The concept of dynamic analysis,” Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp.216–234, 1999.

[7] D. Marinov and R. O’Callahan, “Object equality profiling,” SIGPLAN Not., vol.38, no.11, pp.313–325, oct 2003.

[8] R.P. Buse and W.R. Weimer, “A metric for software readability,” Proceedings of the 2008 International Symposium on Software Testing and Analysis, pp.121–130, 2008.

[9] 丸山僚太，矢口博之，竹下直幸，水野　昭，八杉淳一，“フォントの視認性に関する客観的評価法の提案と主観評価との関連について,” 日本人間工学会大会講演集，vol.48spl，pp.238–239，2012.

[10] 佐々木唯，肥後芳樹，楠本真二，“プログラム文の並べ替えに基づくソースコードの可読性向上の試み,” 情報処理学会論文誌，vol.55，no.2，pp.939–946，feb 2014.

[11] W. Dubay, “The principles of readability,” CA, vol.92627949, pp.631–3309, 01 2004.

[12] N. Truong, P. Roe, and P. Bancroft, “Static analysis of students’ java programs,” Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, pp.317–325, 2004.

[13] W.R. Bush, J.D. Pincus, and D.J. Sielaff, “A static analyzer for finding dynamic programming errors,” Softw. Pract. Exper., vol.30, no.7, pp.775–802, 2000.

[14] 大須賀俊憲，小林隆志，渥美紀寿，間瀬順一，山本晋一郎，鈴村延保，阿草清滋，“Cx-checker：柔軟にカスタマイズ可能な c 言語プログラムのコーディングチェッカ,” 情報処理学会論文誌，vol.53，no.2，pp.590–600，2012.

[15] M.C. Linn, K.D. Sloane, and M.J. Clancy, “Ideal and actual outcomes from precollege pascal instruction,” Journal of Research in Science Teaching, vol.24, no.5, pp.467–490, 1987.

[16] S. Fakhoury, D. Roy, A. Hassan, and V. Arnaoudova, “Improving source code readability: Theory and practice,” Proceedings of the 2019 IEEE/ACM 27th International Conference on Program Comprehension, pp.2–12, 2019.

[17] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, “Exploring the influence of identifier names on code quality: An empirical study,” Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering, pp.156–165, 2010.