特別研究報告

題目

Ave:
プログラミング教育におけるコード品質改善のための実績可視化システム

指導教員

楠本 真二 教授

報告者

華山 魁生

平成 31 年 2 月 12 日

大阪大学 基礎工学部 情報科学科

平成 30 年度 特別研究報告

Ave:
プログラミング教育におけるコード品質改善のための
実績可視化システム

華山 魁生

内容梗概

CI ツールなどを用いた自動テストに基づき，プログラムの正しさに焦点を置いた教育はしばしば行われている．しかし，自動テストはプログラムの振る舞いを確認するだけに留まるため，プログラムの品質に焦点を置いた教育を行うことは難しい．

プログラムの品質を客観的に計測するツールは数多く開発されているものの，学生にとっては計測結果があまりに細かく，それらのツールを上手く使いこなすことができない．また，このような計測結果は線引きがなされておらず，品質の高さに基準が定められていないため，学生は自身の作成したプログラムの品質がどれほどの水準を達成しているのか判断できない．結果として，プログラムの品質向上には繋がっていないのが現状である．

そこで本研究では，家庭用ゲームで用いられている「実績」と呼ばれるコンセプトを取り入れ，プログラムの品質を可視化する教育手法を提案する．適用実験として，基礎工学部情報科学科の 3 年生を対象とした実際の授業にこの手法を取り入れ，受講生の作成するプログラムの品質向上，品質に対する受講生の意識改善，およびプログラミングに対するモチベーション向上に効果があることを確認した．

主な用語

プログラミング教育，CI ツール，自動テスト，プログラム品質，実績，可視化

目次

図目次

表目次

1　はじめに

プログラミング教育の中には，自動テストに基づく教育手法が存在する．この教育手法では，学生が提出したプログラムに対してビルド/テストを自動で行い，教員が事前に定めたテストを通過したか否かで評価を行う．以降ではこのような形態で行われる教育を**テストベースの教育**と呼ぶ．テストベースの教育では，課題の成否が自動かつ即座に判定可能であるため，プログラミングに対するモチベーションの向上に一定の効果があり [17]，また教員の負担を大きく減らすことができる [10, 24]．テストベースの教育に対する研究も盛んに行われており [12]，教育手法として一定の効果があることが明らかになっている．

しかし，テストベースの教育では学生が作成したプログラムの外的振る舞いに主眼を置くことが多く，プログラムの内的構造の良さ，すなわちプログラムの品質は軽視される傾向にある [7]．従って，機能拡張と提出を繰り返してプログラムを完成させていくような類の授業では，次第にプログラムの品質が低下していく [22]．さらにテストベースの教育に対する研究においても，プログラムの正しさに焦点を置いた研究は多く存在する [1, 4] が，品質に焦点を置いた研究は少ないと指摘されている [13]．プログラミングの基礎を教える授業ではなく，より実践的なプログラムを開発するような授業では，プログラムの品質について考える必要があり，またそれを考えるような意識付けが必要である．

プログラムの品質を測るツール自体は活発に開発されており，PMD[*1]や Checkstyle[*2]などが広く知られている．これらのツールはソースコードを静的に解析し，バグの温床となる潜在的な問題を検出することや，コードメトリクスを計測することでプログラムの品質を客観的に示すことが可能である．また，Jenkins[*3]に代表される CI ツールとの相性も良く，プログラムのビルド/テストから品質計測までを自動で実施できる．

しかし，品質計測ツールは主に実務開発者が使用することを想定しているため [23]，その計測結果は学生にとってあまりに粒度が細かい．例えば PMD には 300 を超えるルールセットが存在し，コード行数やサイクロマチック数の計測，あるいは「不必要な変数が存在している」といった単純な問題の検出から，クラスの依存関係といった高度な事柄まで，幅広い項目を扱うことができる．その計測結果は細密かつ膨大な量になり，実務開発者には好ましい一方，学生はそれを活用することができず，またプログラミングに対する苦手意識を植え付けかねない．

さらにメトリクスの計測を行う場合，品質の高さに基準を設けるという難しさがある．例えば「`main` メソッドのサイクロマチック数：8」という結果が得られたとしても，この計測結果には線引きがなされていないため，学生は自身の作成したプログラムの品質が，果たしてどれほどの水準を達成している

[*1] `https://pmd.github.io/`
[*2] `http://checkstyle.sourceforge.net/`
[*3] `https://jenkins.io/`

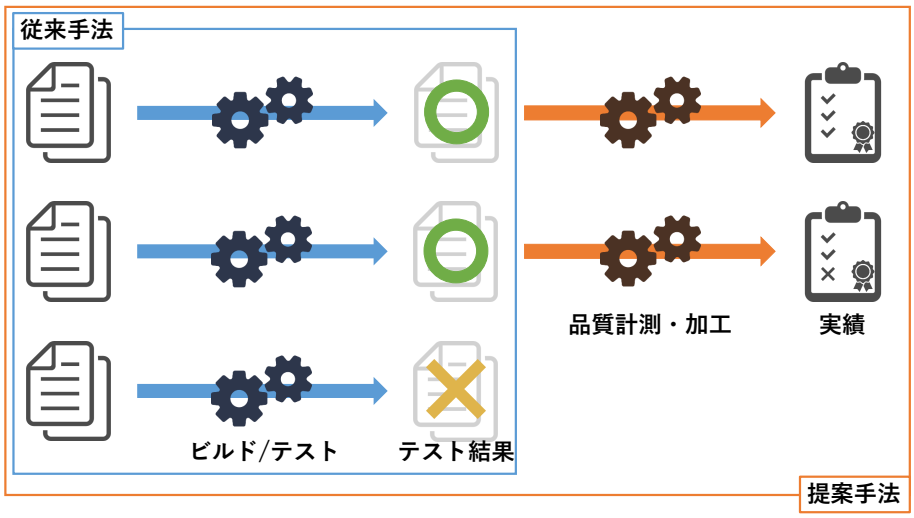

図 1　従来手法と提案手法の概要

のか判断できない．以上の理由から，品質計測ツールをそのままテストベースの教育に組み込むことは難しいと言える．

そこで本研究では，先程までに述べた問題点の解決を目的として，家庭用ゲームで用いられている「実績」と呼ばれるコンセプトを導入し，以下の要件を全て満たすような教育手法を提案する．

- 提出されたプログラムに対して自動でテストを行う
- 提出されたプログラムの品質計測を自動で行う
- 学生にもわかりやすい形で品質の測定結果を提示する

従来手法と本提案手法の概要を図 1 に示す．本提案手法では，テストベースの教育の流れを踏まえた上で，さらにテストを通過したプログラムに対しては品質の計測も行う．計測結果は実績に加工し，それを学生に提示することで，プログラムの品質を簡潔に示すことが可能である．さらに，品質計測までを含めこれらは全て自動で行われるため，テストベースの教育へ容易に組み込むことができる．

この手法を実現するためのプロトタイプのシステムとして Ave を実装し，基礎工学部情報科学科の 3 年生を対象とした実際の授業に導入した．その結果，いくつかの品質項目における改善，プログラムの品質に対する受講生の意識向上，およびプログラミングに対するモチベーション向上が認められた．

以降，2 章では準備として，本研究の提案手法に必要な事柄について説明する．3 章では提案手法について説明し，4 章ではその提案手法を実装したプロトタイプのシステムである Ave について述べる．5 章では Ave の適用実験と結果について述べ，6 章でその結果を考察する．7 章では本研究の妥当性の脅威について述べ，8 章では関連研究について述べる．最後に，9 章で本研究のまとめと今後の課題・展望について述べる．

2 準備

2.1 ゲーミフィケーションとプログラミング教育

ゲーミフィケーションとは，コンピュータ・ゲームの中で特徴的に培われてきたノウハウを現実の社会活動に応用する活動や取り組みのことを指す [25]．ゲーミフィケーションの例としては，既存のシステムに対しランキングや実績，ポイントといった要素を追加することが挙げられる．また，大学のオリエンテーションや写真共有サービスなど，社会生活の様々な場面やサービスにゲーミフィケーションを導入し，どのような効果が得られるのか調査した研究も多く存在する [9, 11, 18]．

さらに，ゲーミフィケーションをプログラミング教育に取り入れるための枠組みを考察している研究 [14] や，実際のプログラミングの授業にゲーミフィケーションを取り入れて検証を行った研究 [3, 5, 15, 21] なども行われている．これらの研究では，ゲーミフィケーションの導入は授業に対する動機づけや受講生の成績上昇に一定の効果があることが示されている．

2.2 バージョン管理システムと CI

バージョン管理システムとは，コンピュータ上で作成・編集されるファイルの変更履歴を管理するためのシステムを指す．このシステムを用いることで，編集箇所の差分を表示することや，あるいは一度編集したファイルを過去の状態に戻すことなどができる．バージョン管理システムの例としては Git[*4] や Apache Subversion[*5] が挙げられる．

CI とは継続的インテグレーション（Continuous Integration）の略称であり，コーディングとビルド/テストを反復して実行していくことで，継続的にプログラムを開発することを意味する．これにより，小さなサイクルで開発を繰り返し行い，エラーを素早く修正することでソフトウェアをより迅速に開発することが可能となる．また，バージョン管理システムとの連携によって，コードのプッシュから CI ツールによるビルド/テストまでを自動で実施できる．CI をサポートするためのツールの例としては，Jenkins などが挙げられる．

2.3 テストベースの教育

本研究では，バージョン管理システムや CI ツールなどの援用により，学生が提出したプログラムに対してビルド/テストを自動で行い，教員が事前に定めたテストを通過したか否かで評価を行う教育手法を**テストベースの教育**と呼ぶ．テストベースの教育における授業の流れは以下の通りである．

[*4] https://git-scm.com/

[*5] http://subversion.apache.org/

1. 教員が課題となるプログラムの仕様を策定する
2. プログラムの振る舞いをチェックするテストを登録する
3. 学生が仕様を満たすプログラムを作成する
4. 作成したプログラムをテストする
5. プログラムが全テストを通過すれば課題を達成したとみなされる

テストベースの教育では，プログラムの振る舞いが正しいか否かの評価を自動かつ即座に行うことができ，プログラミングに対するモチベーション向上にも一定の効果がある [17]．また，教員は学生の作成したプログラムの全てに目を通す必要がなくなるため，テストベースの教育を導入することで教員の負担を大きく減らすことができる [10, 24]．

2.4 テストベースの教育における課題

テストベースの教育にはいくつかの利点がある一方，欠点も存在する．特にテストベースの教育の重大な欠点として，プログラムの外的振る舞いが正しいか，間違っているかといった二値的なフィードバックしか与えず，プログラムの内的構造の良さを軽視していることが挙げられる [7]．ここでの内的構造の良さとは，例えば「各メソッドが短く簡潔である」「不必要な変数が存在しない」といった，コードの可読性や保守性に直結する事柄を指す．以降では，この内的構造の良さを単に**プログラムの品質**と呼ぶ．

プログラミング言語の基本文法や，データ構造とアルゴリズムといった，プログラミングを行う上で土台となる基礎的な知識を教える授業では，プログラムの品質までを取り上げる必要はない．しかし，それらを身に着けた上でより実践的なプログラムを開発するような授業では，プログラムの品質を考える必要があり，またそれを考えるような意識付けが必要である．

プログラムの品質を測るツールは数多く開発されており，PMD や Checkstyle などが広く知られている．これらのツールはソースコードを静的に解析し，プログラムの品質を客観的に示すことが可能である．また，Jenkins に代表される CI ツールとの相性も良く，品質計測ツールと CI ツールを組みわせることで，学生の提出したプログラムをビルド/テストした後に自動で品質計測を実施できる．

しかし，品質計測ツールによる計測結果は非常に粒度が細かく [23]，学生にそのような結果を提示してもそれを活用することができない．また，ツールによる計測結果は線引きがなされておらず，品質の高さに基準が定められていないため，学生は自身の作成したプログラムの品質がどれほどの水準を達成しているのか判断できない．以上のような理由から，品質計測ツールをそのままテストベースの教育に導入することは難しいと言える．

3　提案手法

3.1　実績の導入

本研究の目的は，テストベースの教育においてプログラムの品質を取り上げ，学生の作成するプログラムの品質を向上させること，プログラムの品質に対する学生の意識を改善すること，およびプログラミングに対するモチベーションを向上させることである．本研究ではこの目的を達成するために，テストベースの教育に**実績**と呼ばれるコンセプトを取り入れることを提案する．実績とは家庭用ゲームで用いられている仕組みであり，一定の基準を満たすことによって得られる証や業績を示すものである．本研究ではプログラムの品質の高さを特徴づける項目を実績として設定し，テストベースの教育に取り入れる．

実績を取り入れることにより，品質の高さに基準を設けて線引きを行い，品質が高いとはどういうことか，どうすれば品質が高くなるのかを学生にも理解しやすい形で示すことができる．また，ゲーミフィケーションの要素をプログラミング教育に導入することで，プログラムの品質に対する学生の意識改善とモチベーション向上を促すことができる．

3.2　実績の内容

実績項目を策定するにあたり，本研究で対象とするプログラムの品質を 2 種類に大別し，以下のような実績項目とした．

- **メトリクスベースの実績**：「コード行数」「サイクロマチック数」など定量的な計測を行う項目に基づく実績
- **ルールベースの実績**：「空の `if` 文が存在」「不必要な変数が存在」など二値的な計測を行う項目に基づく実績

メトリクスベースの実績には，閾値の異なる複数の難易度を設定し，段階を設ける．例えば「各メソッドの最大コード行数が規定値以下」という実績を考える．この実績は，各メソッド内の処理が小さく単純化されていることで，読み手に伝わりやすく，また保守性やテストの容易性が上がるという観点でプログラムの品質の高さを表している．この実績に対し，「Lv.1：50 行以下」，「Lv.2：30 行以下」，「Lv.3：10 行以下」といった段階を設けることで，プログラムの品質に基準を設け，理解を深めさせる．また，Lv.1 を比較的容易な難易度に設定することで，実装時の意識付けとモチベーション向上に役立てる．

実績項目やその閾値は，適用する授業内での課題の性質に応じて調整する必要がある．例えば，

Visitor パターン[*6]に則ってプログラムを開発することを想定している課題では，各クラス内に多数のコールバックメソッドを追加する必要がある．この場合「クラスあたりのメソッド数が規定値以下」という実績は，この課題の実績として適していない．

3.3 実績の可視化と比較

本提案手法では，実績の達成状況を可視化すること，および他者と比較することを考える．テストベースの教育では達成すべき課題，あるいは作成すべきプログラムの仕様が，テストという形で教員によって厳密に定められている．学生はその課題を達成するようなプログラムを各自で実装するため，その実装方法はそれぞれ異なるものになる．

そこで，実績の達成状況を可視化し一目で判断できるようにすることで，プログラムの品質の高さを簡潔に示すことができる．また，同じ授業を受講している他の学生と実績の達成状況を比較することで，学生の競争心をあおり，プログラミングに対するモチベーションを向上させることが可能である．

[*6] デザインパターン [8] の一つ

4　実装システム：Ave

4.1　概要

本研究では，提案手法を実現するためのプロトタイプのシステムとして **Ave** を実装した．Ave とは，*Achievements Visualization in programming Education* の略称であり，プログラムの品質を実績として可視化するシステムである．

本提案手法ではプログラムのビルド/テストや品質計測を自動で行うために，表 1 に示すいくつかの外部ツールを使用した．学生の作成したプログラムの品質を計測するツールには PMD，Checkstyle，および Format Feature Extractor[20]（以降，FFE）を用いた．FFE は Java で書かれたソースコードのコーディングスタイルの一貫性を計測するツールである．「代入文の `"="` の後の空白」「`while` 文開始の波括弧（`"{"`）前の空白」などのスタイル特徴を検出し，同一ファイル内でコーディングスタイルが一貫しているかを計測することができる．

4.2　Ave を用いた授業の流れ

テストベースの教育において，従来手法による授業の流れと提案手法による授業の流れを図 2 に示す．なお，外部ツールとしては表 1 に示したツールを用いている．従来手法による授業の流れは以下の手順 1. から手順 5. に示す通りである．なお，リストの番号は図中の番号と一致している．

1. 教員が課題となるプログラムの仕様を策定し，Jenkins 上で Gradle[*7] と JUnit 5[*8] を用いてプログラムの振る舞いをチェックするテストを事前に登録する．
2. 学生は仕様を満たすプログラムを作成したのち，GitBucket[*9] にコードをプッシュする．

表 1　使用した外部ツール

ツールの種類	使用ツール
CI ツール	Jenkins
ビルド/テストツール	Gradle，JUnit 5
Git プラットフォーム	GitBucket
品質計測ツール	PMD，Checkstyle，FFE

[*7] `https://gradle.org/`
[*8] `https://junit.org/junit5/`
[*9] `https://github.com/gitbucket/gitbucket`

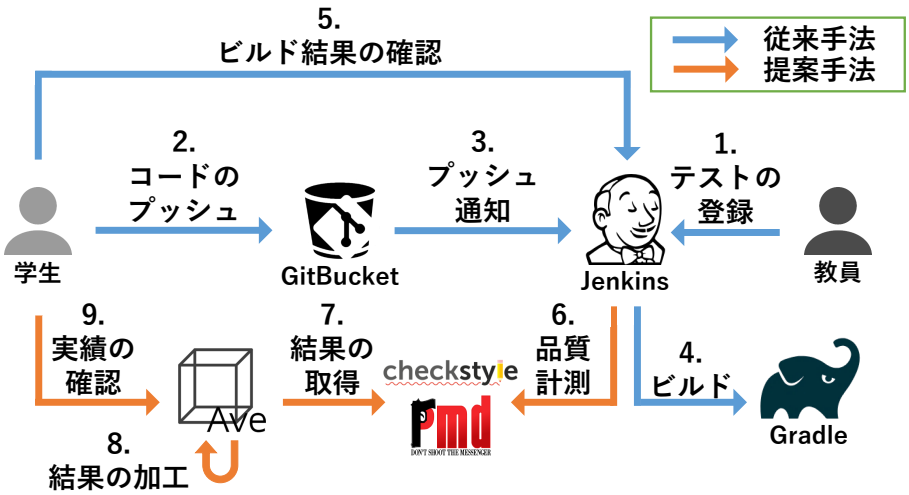

図 2　従来手法と提案手法における授業の流れ

3. GitBucket はコードがプッシュされたことを Jenkins に通知し，Jenkins は GitBucket からコードをプルする．
4. Jenkins は Gradle を用いてプログラムのビルド/テストを行う．
5. 学生が Jenkins 上でビルド結果を確認する．

本提案手法では，テストを通過したプログラムを対象として，追加で以下の手順 6. から手順 9. までの操作も行う．

6. Jenkins が品質計測ツールを用いて品質の計測を行う．
7. Ave が品質計測ツールの計測結果を取得する．
8. 取得した計測結果を実績に加工する．
9. 学生は自身の作成したプログラムがどの実績を達成しているか確認する．

4.3　Ave の各画面

Ave は以下 4 つの画面で構成される．

- 実績達成状況の確認画面
- 実績達成割合の比較画面
- 品質測定結果の確認画面
- API ドキュメント

実績達成状況の確認画面を図 3 に示す．この画面では，課題達成の成否，各実績の概要と達成状況，

実績の達成数を確認することができる．チェックマーク（“✓”）がついているボタンは達成済みの実績，エクスクラメーションマーク（“！”）がついているボタンは未達成の実績を表している．このように，実績の達成状況を可視化し一目で判断できるようにすることで，プログラムの品質の高さを端的に伝えることが可能となる．また，画面中央の上半分がメトリクスベースの実績，下半分がルールベースの実績を表している．メトリクスベースの実績は複数の段階が設けられており，各行が「各メソッドの最大コード行数が規定値以下」といった実績項目，各列が左から Lv.1，Lv.2，Lv.3 の段階を表している．ルールベースの実績は単発的な実績であり，各ボタンがそれぞれの実績項目を表している．

各実績を表すボタンにマウスオーバーすると，その実績の概要が表示される．また，ボタンをクリックすると，より詳細な内容が確認できる API ドキュメント[*10]にジャンプする．この API ドキュメントは，Ave の実装と並行して執筆したものであり，使用している外部ツールの説明，コード例，実績を達成することによる利点，プログラミングにまつわるコラムなどを閲覧することができる．これにより，プログラムの品質に関する様々な知識を提供することができ，またプログラムの品質に対する学生の意識付けを行うことが可能となる．

実績達成割合の比較画面を図 4 に示す．この画面では実績の一覧と，全学生のうち何 % がそれらの実績を達成しているのかを確認することができる．また，緑色の棒グラフで示されている項目は自身が達成済みの実績，灰色の棒グラフで示されている項目は自身が未達成の実績を表している．このように，同じ授業を受講している他の学生と実績の達成状況を比較することで，学生の競争心をあおり，プログラミングに対するモチベーションを向上させることが可能となる．

[*10] https://loki.ics.es.osaka-u.ac.jp/ave/api/

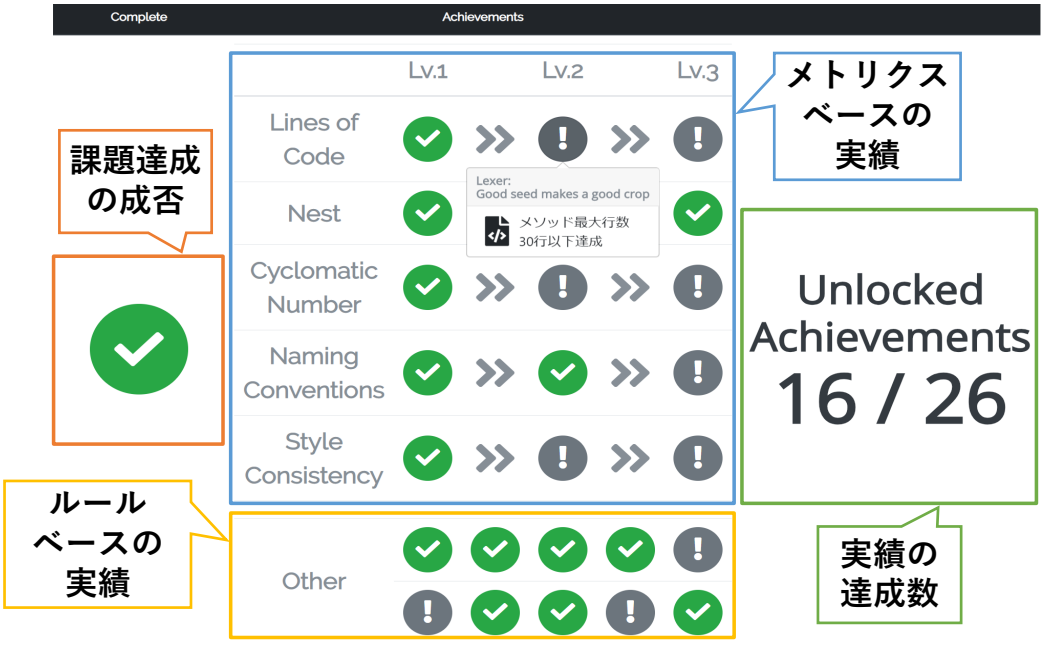

図 3　実績達成状況の確認画面

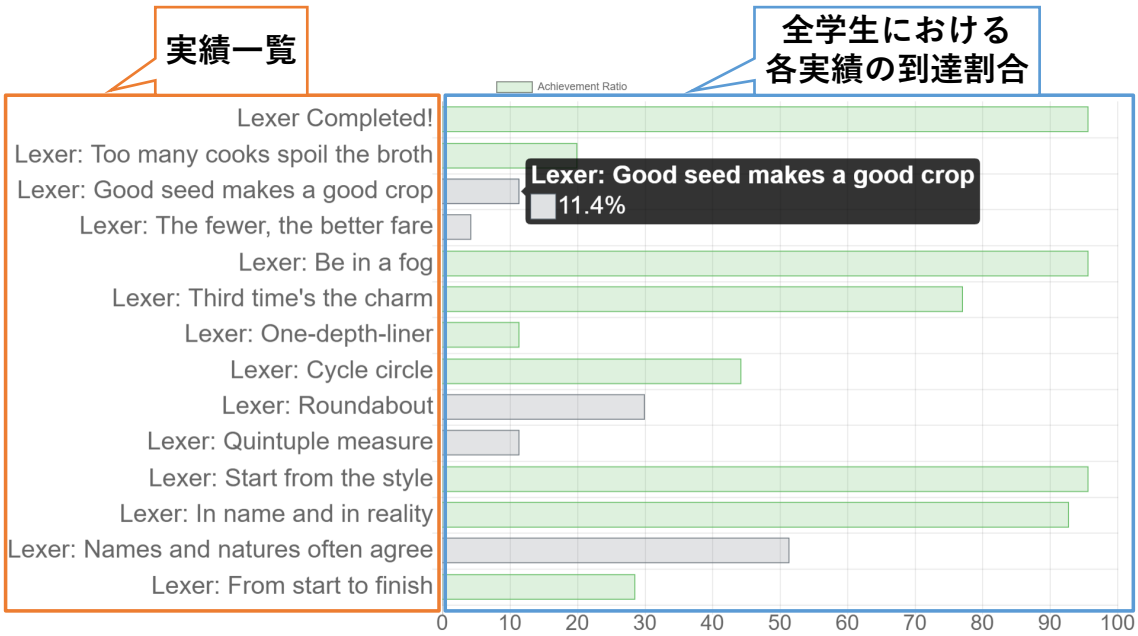

図 4　実績達成割合の比較画面

5 適用実験

5.1 実験の目的

本研究では，Ave をテストベースの教育に導入することでどのような効果が得られるかを確かめるために，適用実験を行った．適用実験を行うにあたり，以下に示す項目をその目的として設定した．

$Q1$： Ave は実際に使用されるか
$Q2$： 受講生の作成するプログラムは品質が向上するか
$Q3$： 品質に対する受講生の意識は改善されるか
$Q4$： プログラミングに対する受講生のモチベーションは向上するか

5.2 適用対象

Ave の適用実験として，基礎工学部情報科学科の 3 年生を対象として開講されている情報科学演習 D[*11]（以降，演習 D）という授業に Ave を取り入れた．演習 D では Java を用いて，Pascal を模した言語で記述されたプログラムをアセンブラ言語 CASL Ⅱで記述されたプログラムに変換するコンパイラを作成する．具体的には，演習 D は字句解析器（Lexer）の作成，構文解析器（Parser）の作成，意味解析器（Checker）の作成，コンパイラ（Compiler）の作成という 4 つの課題で構成されている．本論文の執筆現在では課題 3 の意味解析器の作成まで授業が進行しているため，本章では課題 1〜課題 3 への適用の結果を述べる．

Ave は演習 D に導入されたが，演習 D の受講生が Ave を使用するかどうかは任意とした．そこで，演習 D の受講生には初回授業時に同意書を配布し，Ave を使用する場合は研究目的によるデータ収集の同意を得た．結果として，全受講生 77 名中 71 名から同意を得られた．

5.3 採用した実績

採用したメトリクスベースの実績一覧を表 2 に，ルールベースの実績一覧を表 3 に示す．当初は実績項目の候補が多数挙げられていたが，最終的に以下の観点を基準として表 2，表 3 に示す実績項目とした．

- 品質計測ツールを用いて，CI のサイクル内で自動で計測できるメトリクスに基づく実績項目であること
- 学生にとって理解しやすく，かつプログラムの品質という視点で重要な実績項目であること

[*11] https://loki.ics.es.osaka-u.ac.jp/

- 演習 D の「コンパイラを作成する」という課題に即した実績項目であること

5.4 データの収集方法

同意を得られた受講生については，以下のデータを収集した．

- Jenkins のビルドログ
- GitBucket のコミットログ
- 作成したプログラムの品質計測結果

表 2　メトリクスベースの実績一覧

名前	実績の内容
Lines of Code	各メソッドの最大コード行数が規定値以下
Nest	`for` 文，`if` 文，`try` ブロックの各ネストの深さが規定値以下
Cyclomatic Number	各メソッドの最大サイクロマチック数が規定値以下
Naming Conventions	不適切な命名を行っているクラス，メソッド，変数の個数が規定値以下
Style Consistency	コーディングスタイルの一貫性が規定値以上
Coverage	コードカバレッジ（C0 カバレッジ）が一定値以上
Casl	特定の題材における出力 CASL ファイルの行数の総和が一定値以下

表 3　ルールベースの実績一覧

名前	実績の内容
Switch	`switch` 文を適切に用いている
Useless	コード中に使用していない要素や不必要な要素が存在しない
Empty	コード中に内容のないブロックや括弧などが存在しない
Simplify	簡潔なコードが記述されている
Finalize	変更が行われていない変数の宣言時に `final` 宣言を行っている
Declaration	変数，メソッド，クラスなどの適切な宣言を行っている
Import	適切にインポートを行っている
Indentation	ソースコードのインデントに一貫性がある
Magic Number	ソースコード内にマジックナンバーがない
Null	適切に `Null` や `Equals` のチェックを行っている

- Ave 個人ページのアクセスログ
- API ドキュメントのアクセスログ
- 実績達成割合

なお，Ave 個人ページと API ドキュメントのアクセスログについては，Google Analytics[*12]を用いて解析を行った．

また，第 11 回目の授業（2018/12/20）にて Web 上で回答を行うアンケートを配布し，回答してもらうよう受講生に呼び掛けた．アンケートでの質問項目を表 4，表 5 に示す．アンケートの回答形式として，「記述式で回答するもの（記述式）」「あらかじめ用意した選択肢から選択するもの（選択式）」「1〜5 の尺度から当てはまるものを選ぶもの（5 段階評価）」を設定した．また，アンケートには Ave 全般や演習 D 全般について訪ねた質問項目（表 4）と，Ave 導入によって自身にどのような効果が表れたのかを 5 段階で評価してもらう質問項目（表 5）を設けた．結果として，17 名から回答を得ることができた．

演習 D において各課題プログラムを開発する際には，その課題名と同一の名前を持つクラス内にある `run` メソッドが開発対象となっている．例えば，字句解析器（Lexer）を開発する際には `Lexer.run()` が開発対象である．しかし，各受講生が新たなパッケージ，クラス，メソッドなどを追加することは禁止されていないため，`Lexer` クラスと依存関係にある新たなクラスやメソッドが各受講生によって追加される可能性がある．これに対応するために，Jenkins 上でのビルド開始時に，JVM の起動に `verbose` オプションを使用した．`verbose` オプションを使用することで，クラスが呼び出されるたびにクラスに関する情報が表示されるようになる．これによって，各クラスがどのクラスやメソッドを呼び出しているかを記録し，メトリクス計測時に依存関係を加味してデータを抜き出せるようにした．

5.5 実験結果

収集したデータを用いて，5.1 節に述べた観点で Ave 導入による効果の分析を行った．以降では，「Jenkins によるテスト通過後に，その課題名を持つクラスと同一のパッケージ内に存在するファイルに何らかの変更を加えているコミット」を **FAC** と呼ぶ．例えば，字句解析器のテスト通過後に，`Lexer.java` と同一のパッケージ内に存在するファイルに何らかの変更を加えているコミットは FAC となる．また，FAC とは “For–Ave–Commit” の略称であり，テストを通過しているのにも関わらず変更を加えているため，その変更は Ave の実績を達成するためのものと考えられる．さらに，いずれの課題においても FAC を 10 回以上行った受講生のグループを **Ave 上位グループ**，10 回未満の受講生のグループを **Ave 下位グループ**と呼ぶ．Ave の使用に同意した 71 名の受講生のうち，Ave 上位グループは 6 名であった．

[*12] `https://analytics.google.com/analytics/web/`

5.5.1 $Q1$：Ave の使用状況

各課題における FAC 回数と個人ページ閲覧数別の受講生の人数割合を図 5 に示す．この図から，いずれの課題においても FAC を 1 回以上行っている受講生は全受講生の半数を超えること，および Ave 個人ページを 10 回以上閲覧している受講生も全受講生の半数を超えることがわかる．また，Lexer と

表 4　Ave 全般や演習 D 全般についての質問項目

内容	回答方法
Ave をどれくらいの頻度で使用しましたか？	選択式
Ave を使用してみていかがでしたか？ 率直な感想をお願いします。	記述式
Ave により導入されたコンセプトの中で、良いと思った項目を選んでください。	選択式
1 番目〜3 番目に好きな実績をそれぞれ選んでください。	選択式
Ave を使っている中で分かりにくかった点や、 改善してほしい点があればお教えください。	記述式
「こんな実績が欲しい」と思う実績項目があればお教えください。	記述式
演習 D の他に、Ave と同様のシステムを 導入すると良いと思う授業があればお教えください。	記述式
演習 D は難しかったですか？ 5 段階で評価をお願いします。	5 段階評価
Ave を含め、演習 D に対する全体的なご意見やご感想があればお教えください。	記述式

表 5　Ave 導入による効果を尋ねる質問項目

内容	回答方法
プログラムの品質に関する用語を知ることができた	5 段階評価
プログラムの品質とは具体的に何を指すのか知ることができた	5 段階評価
プログラムの品質を改善する重要性が認識できた	5 段階評価
プログラムの品質を意識しながらプログラムを作成することができた	5 段階評価
テスト通過後に追加で プログラムの品質を向上させるための変更を行うことができた	5 段階評価
自身の作成したプログラムの品質が以前より改善された	5 段階評価
今後、品質の高いプログラムを作成しようと思うきっかけになった	5 段階評価
プログラミングに対するモチベーションが向上した	5 段階評価

Parser に比べて，Checker は FAC を行っている受講生が比較的少ないこともわかる．FAC 回数と個人ページ閲覧数で受講生をソートした上での，各課題における FAC 回数と個人ページ閲覧数の累積比率を図 6 に示す．いずれの課題においても，受講生 71 名の FAC の総和のうち Ave 上位グループ 6 名による FAC の総和が 50% 以上を占めており，特に Lexer では約 69% を占めていた．以上のことから，約半数の受講生が Ave を使用しており，特に上位約 10% の受講生は積極的に Ave を使用していることがわかる．さらに，Ave の使用頻度には受講生間で大きな差があることもわかる．

各課題におけるメトリクスベースの実績達成割合を図 7 に，ルールベースの実績達成割合を図 8 に示す．メトリクスベースの実績ではどの実績も段階が上がるたびに達成割合が低くなっていたが，Naming Conventions は全体的に達成割合が比較的高く，Lines of Code は達成割合が低かった．また，おおむねどの実績でも Parser では達成割合が高くなっていた．ルールベースの実績では，Import や Null の達成割合が比較的高く，Finalize と Magic Number の達成割合が低かった．このように，ルールベースの実績，メトリクスベースの実績ともに実績項目によって達成状況に大きな差が生まれていた．一方で，全実績を達成している受講生も数名いた．この全実績を達成している受講生は，全員が Ave 上位グループであった．すなわち，FAC が多い受講生は他の受講生よりも多くの実績を達成していることがわかる．

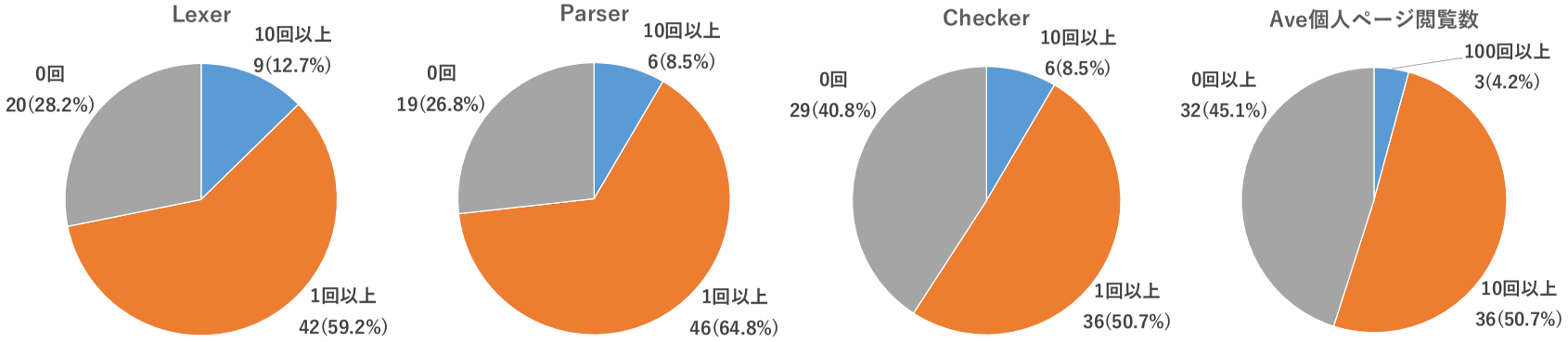

図 5　各課題における FAC 回数と個人ページ閲覧数別の受講生の人数割合

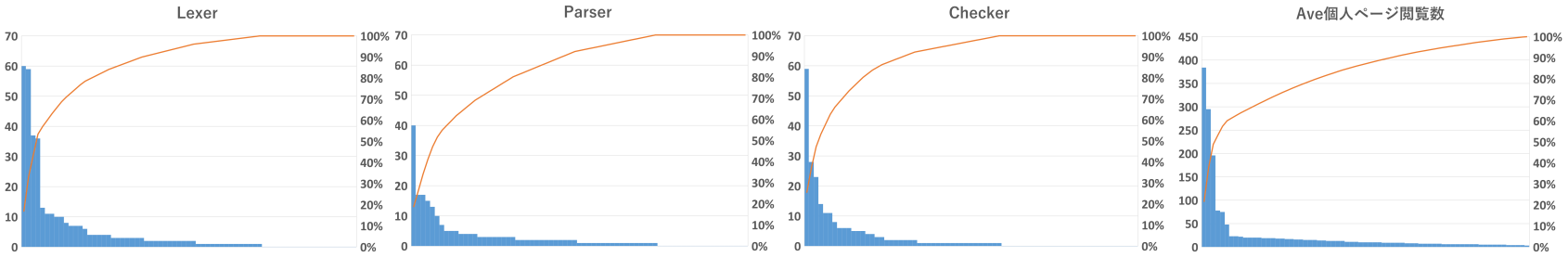

図 6　各課題における FAC 回数と個人ページ閲覧数の累積比率

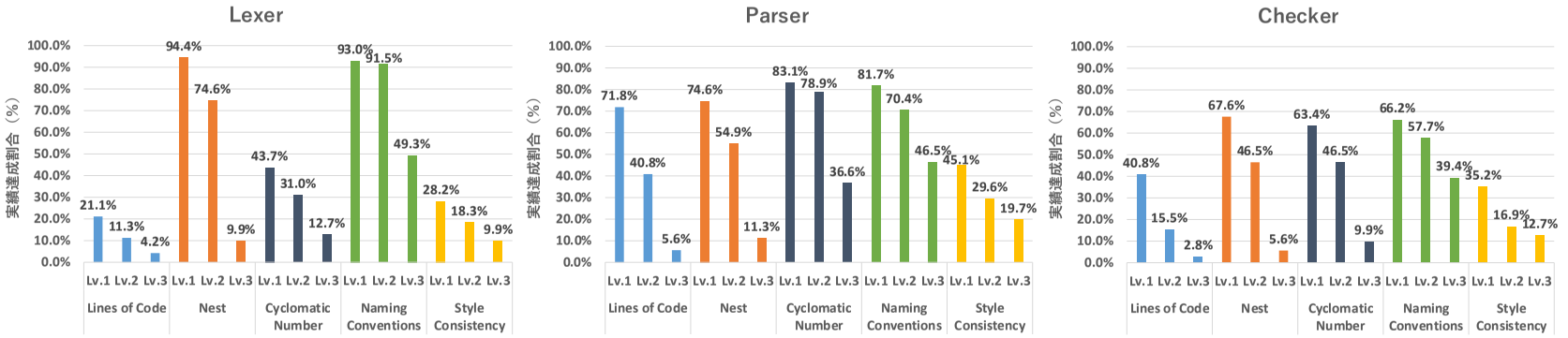

図 7　各課題におけるメトリクスベースの実績達成割合

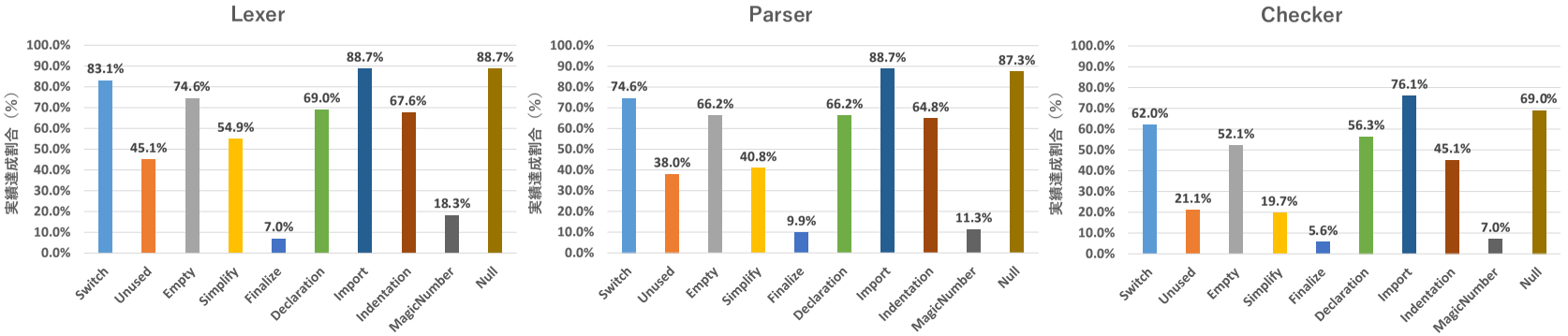

図 8　各課題におけるルールベースの実績達成割合

5.5.2 $Q2$：プログラムの品質向上

5.5.1 項では，受講生間で Ave の使用状況に大きな差があることがわかった．そこで，Ave 上位グループと Ave 下位グループの間で作成するプログラムの品質に差があるかを確かめた．

各課題において，各メソッドあたりの平均コード行数，平均サイクロマチック数，および各クラスあたりの不適切な命名[*13]を行っている（以降，命名規則違反）数の平均を Ave 上位グループと Ave 下位グループで比較した結果を表 6 に示す．表に示されている通り，Ave 下位グループに比べて Ave 上位グループの作成するプログラムは，どの品質項目に関しても改善されていることが確認できる．

5.5.3 $Q3$：品質に対する意識の改善

FAC を一部抜粋したものを図 9，図 10 に示す．図 9 では，コード内で変更が加えられていない変数に `final` が付加されていた．変数に `final` が付加されることで，その変数が不変であることが保証されるのでデバッグしやすくなり，保守性も向上する．図 10 では，処理に `if-then-else` 節を使っていたところを，単に `boolean` 型の値を返すだけの簡潔なコードに変換されていた．不必要な `if-then-else` 節を使う代わりにこのような記述を行うことで，コードの見通しが良くなったりバグを防いだりすることができる．また，図 9 に示す FAC ではコミットメッセージに「**ave 用に final 定義をした**」と記されており，図 10 に示す FAC では「**ave の指摘修正**」と記されていた．その他，受講生の FAC にはコーディングスタイルの統一やメソッド分割などが含まれていた．

5 段階評価によるアンケートの結果を図 11 に示す．このアンケートは 5.4 節で述べた通り，Ave 導入によって自身にどのような効果が表れたのかを 5 段階で評価してもらうものであり，5 が「そう思う」，1 が「そう思わない」である．図に示されている通り，多くの受講生が「今後、品質の高いプログラムを作成しようと思うきっかけになった」「プログラムの品質を改善する重要性が認識できた」などと回答していることがわかる．

表 6　各課題ごとの Ave 上位グループと Ave 下位グループの品質計測結果

内容	Lexer		Parser		Checker	
	上位	下位	上位	下位	上位	下位
平均コード行数 / メソッド	6.77	10.08	6.47	8.79	7.33	9.89
平均サイクロマチック数 / メソッド	2.11	3.47	2.33	3.23	2.67	3.61
平均命名規則違反数 / クラス	0.33	5.20	12.00	14.71	12.17	22.60

[*13] クラス名にパスカルケースを用いていないなど，Java の慣習に反する命名

```
@@ -41,20 +38,20 @@
     ...
-    List<String> lines = new FileInput ...
+    final List<String> lines = new FileInput ...
     ...
```

図 9　final を付加しているコミット

```
@@ -83,11 +83,7 @@
     ...
-    if (checkVariableDeclarationAll()) {
-      return true;
-    } else {
-      return false;
-    }
+    return checkVariableDeclarationAll();
     ...
```

図 10　コードを簡潔にしているコミット

さらにアンケートに対する記述式の回答には，

- 結果が目に見えて分かりやすいので，動くだけでなくより良いコードを書こうという意欲がわいた
- Ave は今までにないシステムで意識してやってみると確かにプログラムの質を考えて設計でき，結果的に今までよりも質が上がったように感じます
- どのようなコードが良しとされるのかを知ることができた

などの記述があり，演習 D 課題レポートの感想欄に，

- 今回はテストが通るようになった後，Ave の実績解除を目標としてコード整理を行っていった．そのため，今までは行っていなかったマジックナンバーの除去や，1 つのメソッドの行数削減などの観点からも考えることができ，後から変更しやすいコードが書けた
- テストが通るようになった後には Ave の実績解除を目標としてコード整理を行っていった．その際，代入しているのがコンストラクタでのみという変数に対しても final をつけることができるというのを知ることができた．他にも，できるだけ簡潔になるように意識したり，マジックナンバーを除去したりと，わかりやすいコードを書く良い機会になったと思う

との記述を行っている受講生もいた（句読点の表記を一部改めた）．以上のことから，Ave 導入によりプログラムの品質に対する意識の改善を複数の受講生に行えたことがわかる．

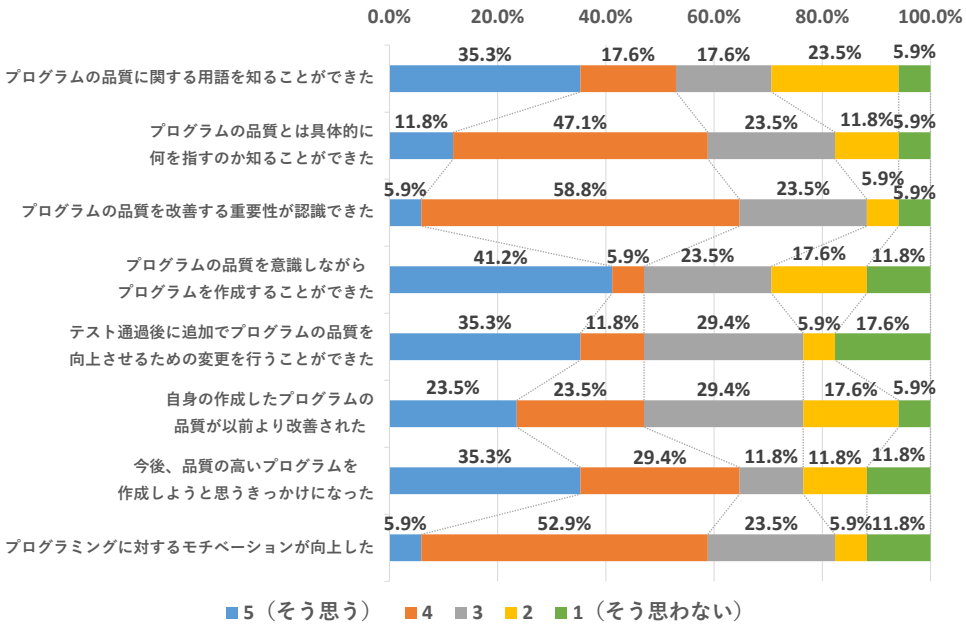

図 11　5 段階評価によるアンケートの結果

5.5.4　$Q4$：プログラミングに対するモチベーション向上

図 11 に示されているように，多くの受講生が「プログラミングに対するモチベーションが向上した」と回答していることがわかる．また，アンケートに対する記述式の回答には，

- Ave によって良いプログラムを作るという効果もあるけど単純に目に見えるとわかりやすいのでモチベーションの向上につながるのがいいところだと思います
- 面白かった．コードの書き方など勉強になった
- コード整理をゲーム感覚でできて楽しかった

などの記述があり，演習 D 課題レポートの感想欄に，

- Ave においては，モチベーションの向上を図ることができ，最終的には提出時点での trial, lexer の実績一覧を全てを解除することができた
- 今回も Ave の達成を頑張った．（中略）ゲーム感覚でコードがより良くなるので，私の性格には大変あっていた．楽しかった

との記述を行っている受講生もいた（句読点の表記を一部改めた）．以上のことから，Ave 導入により複数の受講生がプログラミングに対するモチベーションを向上させたことがわかる．

6　考察

本章では，適用実験の結果について考察する．

$Q1$：Ave は実際に使用されるか

$Q1$ に関する実験の結果から，約半数の受講生が Ave を使用しており，特に上位約 10% の受講生は積極的に Ave を使用していること，また Ave の使用頻度には受講生間で大きな差があることがわかった．

図 5 に関して，Lexer と Parser に比べて Checker において FAC を行っている受講生が比較的少ないことがわかった．これは，Checker や Compiler の難易度が他の課題の難易度に比べて高く，実績を達成するための変更を加える余裕が受講生になかったためと考えられる．

また，図 7 に関して，メトリクスベースの実績ではどの実績も段階が上がるたびに達成割合が低くなっていくが，Naming Conventions は達成割合が比較的高く，Lines of Code は達成割合が低いことがわかった．これは，Naming Conventions はクラス名やメソッド名に変更を加えるだけで達成できる比較的容易な実績である一方，Lines of Code はメソッドの抽出やプログラムフローの改善など，プログラムの内部構造を大きく改善する必要がある比較的難しい実績であるためと考えられる．さらに，おおむねどの実績でも Parser では達成割合が高くなっているが，これは Parser の課題が容易であり，実績を達成するために変更を加える余裕が受講生にあったこと，あるいは Parser の実績の閾値が適切でなかったことなどが考えられる．

ルールベースの実績では，Import や Null の達成割合が比較的高く，Finalize と Magic Number の達成割合が低いことがわかった．これらの実績についてもメトリクスベースの実績と同じく，実績の難易度の差によって達成状況に差が生まれていると考えられる．

$Q2$：受講生の作成するプログラムは品質が向上するか

$Q2$ に関する実験の結果から，Ave 下位グループに比べて Ave 上位グループの作成するプログラムは，どの品質項目に関してもプログラムの品質が改善されていることがわかった．

表 6 に関して，各メソッドあたりの平均コード行数，各メソッドあたりの平均サイクロマチック数，各クラスあたりの命名規則違反数の平均は，いずれも Ave 下位グループに比べて Ave 上位グループの作成するプログラムの方が改善されていた．これにより，Ave の導入がプログラムの品質向上に貢献していることがわかる．ただし，本実験で計測を行ったこれらの品質項目は Ave の実績として設定している項目であり，実績を達成することはこれらの品質項目の改善に直結することに留意が必要である．

$Q3$：品質に対する受講生の意識は改善されるか

$Q3$ に関する実験の結果から，Ave 導入によりプログラムの品質に対する意識の改善を複数の受講生に行えたことがわかった．

図 9 や図 10 に抜粋した FAC をはじめとして，受講生のコミットにはプログラムの品質を向上させるための変更を行っているものがいくつも見られた．また，コミットメッセージに「ave」という文言が含まれていることや，Ave 導入に関するアンケート，演習 D 課題レポートの感想欄における記述からもわかるように，明らかに Ave 導入によって品質に対する受講生の意識が改善されていることがわかる．ただし，図 11 に示すアンケート結果のうち，「自身の作成したプログラムの品質が改善された」「プログラムの品質を意識しながらプログラムを作成することができた」という質問項目に関しては，他の質問項目より「そう思う」と答えた受講生が比較的少ない．このように，実際に自身の作成したプログラムの品質が向上したと考えている受講生はやや少ないものの，本研究の目的の一つとしている品質に対する受講生の意識改善に関しては，Ave 導入の効果があったと考えられる．

$Q4$：プログラミングに対する受講生のモチベーションが向上するか

$Q4$ に関する実験の結果から，Ave 導入により複数の受講生がプログラミングに対するモチベーションを向上させたことがわかった．

図 11 に示されているように，多くの受講生が「プログラミングに対するモチベーションが向上した」と回答しており，アンケートに対する記述式の回答や演習 D 課題レポートの感想欄における記述でも，楽しい・面白い・わかりやすいといった感想が受講生から得られた．Ave をテストベースの教育に取り入れることで，新たにゲーミフィケーションの要素を追加することができ，結果としてプログラミングや授業への参加に対する意欲向上に繋がると考えられる．

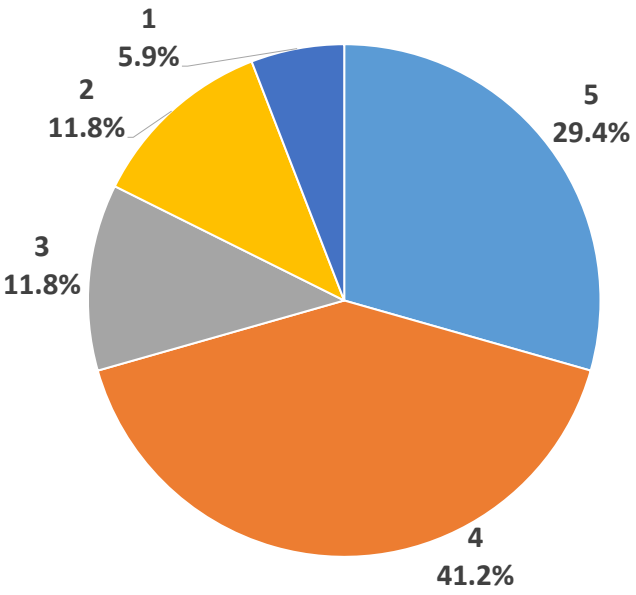

図 12　演習 D の難易度に関するアンケートの結果

7　妥当性への脅威

本研究では，適用実験の対象として演習 D を選択しており，演習 D で設定されている課題の性質に合わせて実績項目やその閾値が策定された．そのため，実績項目やその閾値を変更すると，調査結果が変わる可能性がある．また，Ave を導入する対象を他の授業に変える際には，実績項目やその閾値を適用対象に合わせて変更する必要があり，調査結果を比較することが難しい．

演習 D そのものの難易度が高く，実績の達成にまで手が回らなかった受講生も多い．図 12 に「演習 D は難しかったですか？ 5 段階で評価をお願いします。」というアンケートの結果を示す．5 が「難しかった」，1 が「簡単だった」である．このように大半の受講生が演習 D は難しかったと回答しており，プログラムの品質を考えたり Ave の実績を達成するために変更を行ったりする余裕がなかったため，得られたデータが偏っている可能性がある．

加えて，本研究においてプログラムの品質とは，「各メソッドが短く簡潔である」「不必要な変数が存在しない」といった，コードの可読性や保守性に直結する事柄としている．しかし，プログラムの品質モデルとしては，ISO/IEC25010[*14]という国際規格が定められている．本研究の提案手法により改善がみられたプログラムの品質とは，この国際規格で定められたプログラムの品質の中の一部であることに注意が必要である．

[*14] https://www.iso.org/standard/35733.html

8　関連研究

テストベースの教育やゲーミフィケーションに関する研究は盛んに行われている [6, 11, 12]．本章ではそれらの研究のうちいくつかを説明し，本研究との関連や差異について述べる．

Kyrilov らの研究

Kyrilov らは，テストベースの教育について研究を行った [16]．この研究では学部生向けに開講されているプログラミングの授業を対象としており，連続する授業のうちいくつかの授業で，受講生は提出したプログラムの振る舞いが正しいのか，間違っているのかといった二値的な結果だけを受け取った．調査の結果，二値的な結果を受け取った受講生は受け取らなかった受講生と比べて，約 2 倍も不正行為を行いやすくなることがわかった．また，二値的な結果を受け取った受講生は，授業への参加率が低下することもわかった．結論として，フィードバックが二値的なものに限られているシステムをプログラミング教育に導入することは，受講生に望ましくない行動をさせる可能性があり，正しいか間違っているかといった二値的な結果に加えて，より多様なフィードバックを返す必要があるとしている．

本研究ではこのような研究を動機として，二値的な結果に加えて実績という形による品質計測結果もフィードバックしており，品質に対する受講生の意識改善やプログラミングに対するモチベーション向上に貢献することができた．

Barata らの研究

Barata らは，講義形式の授業にゲーミフィケーションの要素を取り入れることにより，どのような効果が得られるのかを検証した [2]．この研究では，大学院生向けに開講されている授業に経験値，レベル，実績などを用いてゲーミフィケーションの要素を取り入れた．研究の結果は非常に良好なものであり，ゲーミフィケーションの要素を取り入れなかった年度に比べて，取り入れた年度では受講生の出席率や教材の閲覧数の大幅な上昇がみられた．また，受講生はゲーミフィケーションの要素が授業に導入されたことにより，モチベーション向上や授業の理解度上昇に繋がったと回答していた．結論として，ゲーミフィケーションの要素を授業に取り入れることで非常に良い効果がもたらされるとしている．

本研究との大きな差異として，Barata らの研究対象と本研究の対象では授業形態が異なることが挙げられる．Barata らの研究で対象とした授業は講義形式の授業であり，本研究ではテストベースの教育を対象としている．また，その適用実験として演習形式で開講されている演習 D に提案手法を導入した．その結果，演習形式の授業であってもゲーミフィケーションの要素を取り入れることは有効であることを示した．

O'Donovan らの研究

O'Donovan らは，演習形式の授業にゲーミフィケーションの要素を取り入れる研究を行った [19]．この研究では，学部生向けに開講されているゲームデザインを学びそれを実装する授業に，ランキング，ストーリー，景品，ゲーム内通貨などを用いてゲーミフィケーションの要素を取り入れた．研究の結果，受講生の出席率の改善，および理解度や問題解決能力の向上がみられた．

しかし，この研究はプログラムの品質に言及しておらず，研究の主目的を受講生の参加態度の改善や理解度の向上としている．本研究では，ゲーミフィケーションの要素を取り入れることで受講生のモチベーション向上だけでなく，受講生の作成したプログラムの品質向上やプログラムの品質に対する受講生の意識改善に効果があることを確認できた．

9　おわりに

本研究では，自動テストに基づくプログラミング教育に，家庭用ゲームで用いられている「実績」と呼ばれるコンセプトを取り入れ，プログラムの品質を可視化する教育手法を提案した．

本提案手法では，自動テストに基づくプログラミング教育の流れを踏まえた上で，テストを通過したプログラムに関しては外部ツールの援用の下で品質の計測を行う．品質計測結果は実績という形に加工し可視化することで，プログラムの品質を学生にもわかりやすい形で示すことが可能である．さらに，実績の達成状況を他の学生と比較することで競争心をあおり，プログラミングに対するモチベーション向上に繋げることができる．

提案手法の適用実験として，基礎工学部情報科学科の 3 年生を対象とした実際の授業にこの手法を取り入れ，プログラムの品質向上と品質に対する受講生の意識改善，およびプログラミングに対するモチベーション向上に効果があることを確認した．

本研究の今後の課題としては，得られたデータを用いて適切な実績項目とその閾値を策定することや，他の授業にも適用できるように柔軟性を向上することが挙げられる．本研究では，実装したプロトタイプのシステムである Ave を演習 D に初めて導入した．従って，実績項目やその閾値にはまだ改善の余地があり，改良を加えることで適用対象の授業に沿ったより適切なシステムになると考えられる．

また現状では，Ave は演習 D に特化した形となっている．しかし，提案手法が前提とする授業形態は「テストベースの教育であること」だけであり，この前提が満たされている授業であれば同様のシステムを構築することができる．従って，他の授業にも適用できるようにするために，Ave の中で演習 D に特化している部分を汎化し，より柔軟性を向上していきたいと考えている．

謝辞

本研究を行うにあたって，理解あるご指導を賜り，励まして頂きました，楠本真二 教授に心より感謝申し上げます．

本研究に関して，使用する外部ツールをはじめ，数々の貴重で有益な助言をして頂きました，肥後芳樹 准教授に深く感謝申し上げます．

本研究の全過程を通し，研究に対する方針や実現方法など，終始丁寧かつ熱心なご指導を賜りました，柗本真佑 助教に心より感謝申し上げます．

本研究の適用実験にご協力頂き，Ave に対する感想や助言，時には暖かい励ましの言葉を頂きました，情報科学演習 D の受講生の皆さん，TA の皆さん，および担当教員の方々に深く御礼申し上げます．

本研究で使用させて頂いた Format Feature Extractor を開発なさいました，楠本研究室の卒業生である小倉直徒 氏に心より感謝申し上げます．

本研究を進めるにあたり，様々な形で励まし，協力を頂きました楠本研究室の皆様のご協力に，心より感謝申し上げます．

本研究に至るまでに，講義，演習，実験等でお世話になりました，大阪大学基礎工学部情報科学科の諸先生方に，この場を借りて心から御礼申し上げます．

最後に大学生活を支え，励ましてくれた家族にも，心より感謝致します．

参考文献

[1] A. Altadmri and N. C.C. Brown. 37 million compilations: Investigating novice programming mistakes in large–scale student data. In *Technical Symposium on Computer Science Education*, pp. 522–527, 2015.

[2] G. Barata, S. Gama, J. Jorge, and D. Goncalves. Engaging engeneering students with gamification. In *International Conference on Games and Virtual Worlds for Serious Applications*, No. October 2016, pp. 1–8, 2013.

[3] K. Berkling and C. Thomas. Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure. In *International Conference on Interactive Collaborative Learning*, pp. 525–530, 2013.

[4] N. C.C. Brown, M. Kölling, D. McCall, and I. Utting. Blackbox: A large scale repository of novice programmers' activity. In *Technical Symposium on Computer Science Education*, pp. 223–228, 2014.

[5] M. Chang and Kinshuk. Web–based multiplayer online role playing game (morpg) for assessing students' java programming knowledge and skills. In *International Conference on Digital Game and Intelligent Toy Enhanced Learning*, pp. 103–107, 2010.

[6] D. Dicheva, C. Dichev, G. Agre, and G. Angelova. Gamification in education: A systematic mapping study. *Systematic Mapping Study. Educational Technology & Society*, Vol. 18, No. 3, pp. 75–88, 2015.

[7] S. H. Edwards. Rethinking computer science education from a test–first perspective. In *SIGPLAN Conference on Object–Oriented Programming, Systems, Languages, and Applications*, pp. 148–155, 2003.

[8] G. Erich, H. Richard, J. Ralph, and V. John. *Design Patterns: Elements of Reusable Object–Oriented Software*. Addison–Wesley, 1994.

[9] Z. Fitz-walter, D. Tjondronegoro, and P. Wyeth. Orientation passport: using gamification to engage university students. In *Australian Computer–Human Interaction Conference*, pp. 122–125, 2011.

[10] J. A. Harris, E. S. Adams, and N. L. Harris. Making program grading easier: but not totally automatic. *Journal of Computing Sciences in Colleges*, Vol. 20, No. 1, pp. 248–261, 2004.

[11] K. Huotari and J. Hamari. Defining gamification – a service marketing perspective. In *International Academic MindTrek Conference*, Vol. 1, pp. 17–22, 2012.

[12] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. *Turkish Journal of Gastroenterology*, Vol. 18, No. 4, pp. 265–267, 2007.

[13] H. Keuning, B. Heeren, and J. Jeuring. Code quality issues in student programs. In *Conference on Innovation and Technology in Computer Science Education*, pp. 110–115, 2017.

[14] F. L. Khaleel, N. S. Ashaari, T. S. Meriam, T. Wook, and A. Ismail. The study of gamification application architecture for programming language course. pp. 1–5, 2015.

[15] A. Knutas, J. Ikonen, U. Nikula, and J. Porras. Increasing collaborative communications in a programming course with gamification. In *International Conference on Computer Systems and Technologies*, pp. 370–377, 2014.

[16] A. Kyrilov and D. C. Noelle. Binary instant feedback on programming exercises can reduce student engagement and promote cheating. In *Koli Calling Conference on Computing Education Research*, pp. 122–126, 2015.

[17] A. Kyrilov and D. C. Noelle. Do students need detailed feedback on programming exercises and can automated assessment systems provide it? *Journal of Computing Sciences in Colleges*, Vol. 31, No. 4, pp. 115–121, 2016.

[18] M. Montola, T. Nummenmaa, A. Lucero, M. Boberg, and H. Korhonen. Applying game achievement systems to enhance user experience in a photo sharing service. In *International MindTrek Conference: Everyday Life in the Ubiquitous Era*, pp. 94–97, 2009.

[19] S. O'Donovan, J. Gain, and P. Marais. A case study in the gamification of a university–level games development course. pp. 242–251, 2013.

[20] N. Ogura, S. Matsumoto, H. Hata, and S. Kusumoto. Bring your own coding style. In *International Conference on Software Analysis, Evolution and Reengineering*, pp. 527–531, 2018.

[21] M. Olsson, P. Mozelius, and J. Collin. Visualisation and gamification of e–learning and programming education. Vol. 13, No. 6, pp. 441–454, 2015.

[22] R. Pettit, J. Homer, R. Gee, S. Mengel, and A. Starbuck. An empirical study of iterative improvement in programming assignments. In *Technical Symposium on Computer Science Education*, pp. 410–415, 2015.

[23] P. Tomas, M. J. Escalona, and M. Mejias. Open source tools for measuring the internal quality of java software products. a survey. *Computer Standards and Interfaces*, Vol. 36, No. 1, pp. 244–255, 2013.

[24] G. Tremblay and E. Labonte. Semi–automatic marking of java programs using junit. In *International Conference on Education and Information Systems: Technologies and Applications*, pp. 42–47, 2003.

[25] 井上明人. ゲーミフィケーション – ＜ゲーム＞がビジネスを変える. NHK 出版, 2012.