社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

プログラミング教育における実績制度を用いたコード品質可視化システムの試作

華山　魁生†　　柗本　真佑†　　肥後　芳樹†　　楠本　真二†

† 大阪大学大学院情報科学研究科
〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †{k-hanaym,shinsuke,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　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，学部 3 年生を対象とした実際の授業にこの手法を取り入れ，学生の作成するプログラムの品質向上と品質に対する学生の意識改善に効果があることを確認した．

キーワード　プログラミング教育，CI ツール，プログラム品質，実績，可視化

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，学生が提出したソースコードに対してビルド/テストを自動で行い，教員が事前に定めたテストに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 [1]，また教員の負担を大きく減らすことができる [2] [3]．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 [4] [5]，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の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 [6]．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ことになる [7]．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 [8] [9] が，品質に焦点を置いた研究は少ないと指摘されている [10]．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計測するツール自体は活発に開発されており，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，バグの温床となる潜在的な問題を検出したり，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できる．また，Jenkins 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行うことができる．

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 [11]，その計測結果は学生にとってあまりに粒度が高い．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：32」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おらず，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．

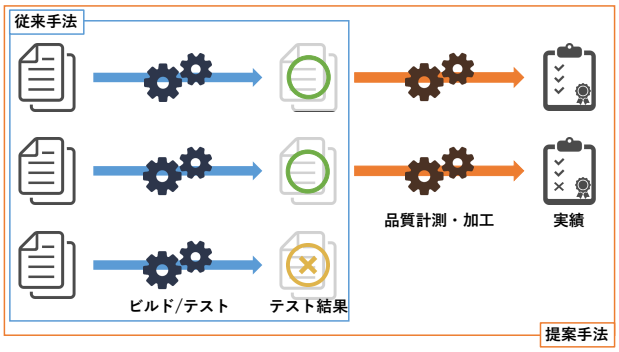

図 1　従来手法と提案手法の概要

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，さらにテストに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できる．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に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として Ave を実装し，学部 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，およびプログラムの品質に対する学生の意識向上が認められた．

2.　テストベースの教育とその課題

2.1　テストベースの教育

バージョン管理システムや CI ツールなどの援用により，学生が提出したソースコードに対してビルド/テストを自動で行い，教員が事前に定めたテストに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．

（1）　教員が課題となるプログラムの仕様を策定
（2）　プログラムの振る舞いをチェックするテストを登録
（3）　学生が仕様を満たすプログラムを作成
（4）　作成したプログラムをテスト
（5）　プログラムが全テストに通過すれば課題達成

テストベースの教育では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 [1]．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 [2] [3]．

2.2　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，内的構造の良さを軽視していることが挙げられる [6]．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(注1)や Checkstyle(注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，プログラムの品質を客観的に示すことができる．また，Jenkins(注3)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行うことができる．

しかし，品質計測ツールによる計測結果は非常に粒度が高く [11]，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である．我々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．

実績制度を取り入れることで，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，実績というゲーミフィケーション(注4)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

(注1)：https://pmd.github.io/
(注2)：http://checkstyle.sourceforge.net/
(注3)：https://jenkins.io/
(注4)：コンピュータ・ゲームのなか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こと [12]

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(注5)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できる．

4. 実装システム：Ave

4.1 概　　要

我々は提案手法を実現するためのプロトタイプとして **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 [14]（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．

(注5)：デザインパターン [13] の一つ

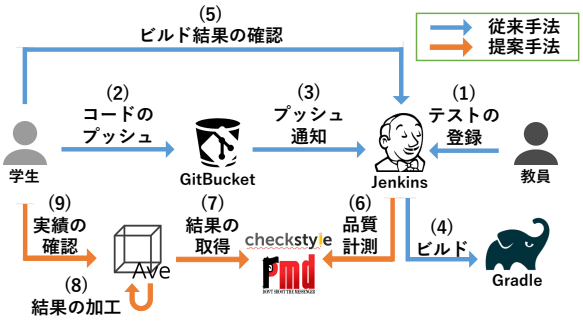

図 2　従来手法と提案手法における授業の流れ

4.2 Ave を用いたテストベースの教育の流れ

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．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(注6) と JUnit 5(注7) 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(注8) にコードをプッシュする．

（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，テストに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

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

(注6)：`https://gradle.org/`
(注7)：`https://junit.org/junit5/`
(注8)：`https://github.com/gitbucket/gitbucket`

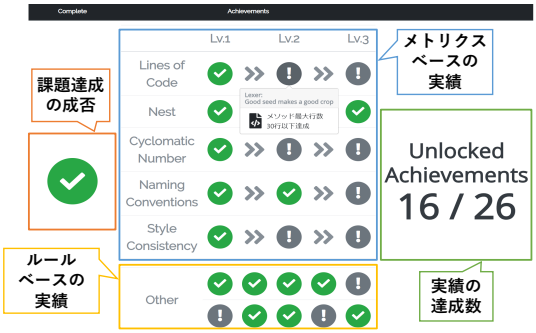

図 3　実績達成状況の確認画面

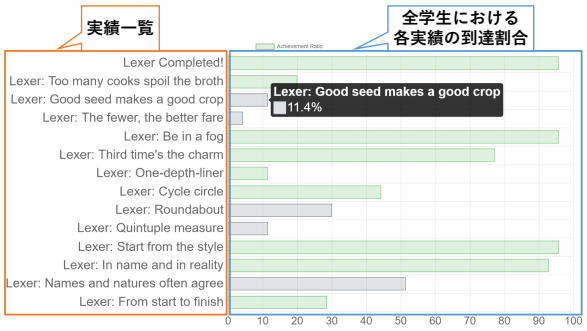

図 4　実績達成割合の比較画面

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できる．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(注9)にジャンプする．この API ドキュメントでは，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，また受講生の意識付けを行うことができる．

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，Ave の利用者のうち何%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受講生と実績の達成状況を比較することで，受講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できる．

5. 適 用 実 験

5.1 適 用 対 象

Ave の適用実験として，大阪大学基礎工学部情報科学科の学部 3 年生を対象として開講されている情報科学演習 D(注10)（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 風言語で記述されたプログラムをアセンブラ言語 CASLII 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，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 1 の字句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 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．

5.2 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．我々は当初挙げられていた実績項目を，以下の観点を基準として絞り込んだ．

- 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
- 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

5.3 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(注11)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

(注9)：`https://loki.ics.es.osaka-u.ac.jp/ave/api/`

(注10)：`https://loki.ics.es.osaka-u.ac.jp/`

(注11)：クラス名にパスカルケースを用いていないなど，慣習に反する命名

- 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
- 実績到達割合

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 を用いて解析を行った．

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，各学生が `Lexer` クラスと依存関係にある新たなクラスやメソッドなどを追加することは禁止されていない．これに対応するために，ビルド時の JVM の起動には `verbose` オプションを使用した．このオプションを使用することで，`Lexer` クラスがどのクラス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．

5.4 実験結果

我々は収集したデータを用いて，以下の観点で Ave 導入による効果の分析を行った．

- Ave は実際に使用されているか
- 受講生の作成するプログラムは品質が向上しているか
- 品質に対する受講生の意識は改善されたか

以降では，「Lexer のテスト通過後に `Lexer` 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．FAC とは “For–Ave–Commit” の略称であり，テストに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．また FAC が 10 以上の受講生グループを **Ave 上位グループ**，FAC が 10 未満の受講生グループを **Ave 下位グループ**と呼ぶ．同意を得られた 71 名の受講生のうち，Ave 上位グループは 7 名であった．なお，全受講生の JVM 起動ログを確認し，どの受講生においても `Lexer` クラスと依存関係にあるクラスは，全て `Lexer` クラスと同一のパッケージに存在していることは確認済みである．

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 / Equals	適切に `Null` や `Equals` のチェックを行っている

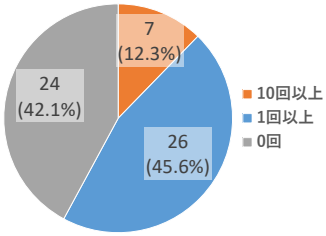

図 5　FAC 回数別の受講生の人数割合

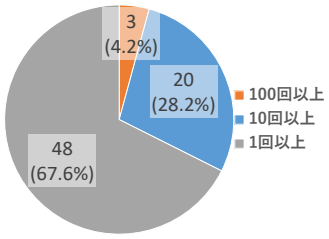

図 6　Ave 個人ページアクセス回数別の受講生の人数割合

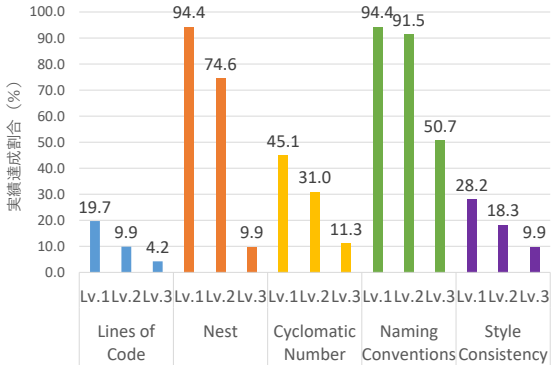

図 7　メトリクスベースの実績達成割合

5.4.1 Ave の使用状況

図 5 に FAC 回数別の受講生の人数割合を示す．図 5 から，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がわかる．また，FAC 回数で受講生をソートしたとき，上位 15 名の FAC 回数の総和が全 FAC 回数の総和の 81.9%を占めていた．図 6 に Ave 個人ページへのアクセス回数の割合を示す．また，FAC 回数と同様に Ave 個人ページへのアクセス回数で受講生をソートしたとき，上位 15 名の受講生のアクセス数の総和が全アクセス数の総和の 73.8%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確認できる．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．

図 7 に各メトリクスベースの実績達成割合を，図 8 に各ルールベースの実績達成割合を示す．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ることがわかる．一方で，全実績を達成している受講生も 2 人いた．

5.4.2 プログラムの品質向上

5.4.1 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，各クラスあたりの不適切な命名を行っている（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 4 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，どのメトリクスに関しても大幅に改善され

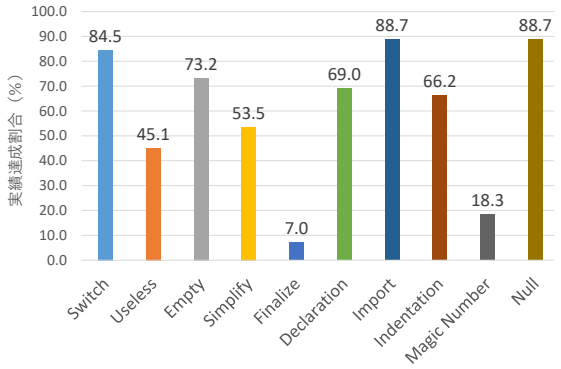

図 8　ルールベースの実績達成割合

```
@@ -20,6 +20,12 @@
  public class Lexer {
+   public static final int IDNUM = 44;
    ...
    if (token.matches("^[0-9]+$")) {
-     id = 44;
+     id = IDNUM;
    ...
  }
```

図 9　マジックナンバーを除去しているコミット

ていることが確認できる．特に命名規則違反に関しては，Ave 上位グループの全員が違反数 0 であった．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．

5.4.3　品質に対する意識の改善

図 9 に FAC を一部抜粋したものを示す．図 9 ではマジックナンバーが除去されている．マジックナンバーとは，ソースコード中にハードコーディングされた数字のことを指す．プログラム内で用いる定数を `final` 宣言された変数にすることにより，数字を変更してしまうことによるバグを回避できる．

また，他の受講生による FAC では，変更が加えられていない変数に `final` が付加されており，コミットメッセージには「**ave 用に final 定義をした**」と記されていた．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，コードの見通しが良くなる．

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おり，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．

6.　おわりに

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した．適用実験として，学部 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学生の意識改善に効果があることを確認した．

表 4　Ave 上位グループと Ave 下位グループの品質計測結果

内容	Ave 下位	Ave 上位
平均コード行数 / メソッド	37.5	8.9
平均サイクロマチック数 / メソッド	10.5	3.2
平均命名規則違反数 / クラス	1.3	0

本研究の今後の課題としては，授業内に設けられた今後の課題に合わせて実績を追加・改良していくことや，受講生にアンケートを行ってさらなるデータ収集を行うこと，授業終了後に全課題のデータを用いて分析を行うことなどが挙げられる．

謝辞　本研究の一部は，日本学術振興会科学研究費補助金基盤研究（B）（課題番号：16H02908，18H03222）の助成を得て行われた．

文　献

[1] 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.
[2] 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.
[3] G. Tremblay and E. Labonte, “Semi–automatic marking of java programs using junit,” International Conference on Education and Information Systems: Technologies and Applications, pp.42–47, 2003.
[4] A. Kyrilov and D.C. Noelle, “Binary instant feedback on programming exercises can reduce student engagement and promote cheating,” Proceedings of the 15th Koli Calling Conference on Computing Education Research, pp.122–126, 2015.
[5] 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.
[6] S.H. Edwards, “Rethinking computer science education from a test-first perspective,” Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp.148–155, 2003.
[7] R. Pettit, J. Homer, R. Gee, S. Mengel, and A. Starbuck, “An empirical study of iterative improvement in programming assignments,” Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp.410–415, 2015.
[8] A. Altadmri and N.C. Brown, “37 million compilations: Investigating novice programming mistakes in large-scale student data,” Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp.522–527, 2015.
[9] N.C.C. Brown, M. Kölling, D. McCall, and I. Utting, “Blackbox: A large scale repository of novice programmers’ activity,” Proceedings of the 45th ACM Technical Symposium on Computer Science Education, pp.223–228, 2014.
[10] H. Keuning, B. Heeren, and J. Jeuring, “Code quality issues in student programs,” Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp.110–115, 2017.
[11] 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.
[12] 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.
[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison–Wesley, 1994.
[14] N. Ogura, S. Matsumoto, H. Hata, and S. Kusumoto, “Bring your own coding style,” 25th IEEE International Conference on Software Analysis, Evolution and Reengineering, pp.527–531, 2018.