情報処理学会論文誌 Vol.61 No.3 644–656 (Mar. 2020)

プログラミング教育における実績可視化システムの提案と評価

華山 魁生[1,a)] 柗本 真佑[1,b)] 肥後 芳樹[1,c)] 楠本 真二[1,d)]

受付日 2019年6月4日, 採録日 2019年11月29日

概要：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舞いを確認するだけにとどまるため，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，これらのツールは問題箇所を指摘するものであり品質の良さを褒めるものではないため，プログラミングに対する苦手意識を学生に植え付ける可能性がある．また，品質計測ツールはメトリクス計測結果を実測値のまま提示する．その結果には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，学生がこれらのツールを使いこなすことは難しく，プログラムの品質向上にはつな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．

キーワード：プログラミング教育，プログラム品質，自動テスト，CI ツール，実績，可視化

Proposal and Evaluation of Achievements Visualization System in Programming Education

Kaisei Hanayama[1,a)] Shinsuke Matsumoto[1,b)] Yoshiki Higo[1,c)] Shinji Kusumoto[1,d)]

Received: June 4, 2019, Accepted: November 29, 2019

Abstract: Programming education focusing on the correctness of program behavior is often conducted with automated testing. However, it is difficult to instruct students about internal program quality since automated testing confirms only the external behavior of the program. Although there are a lot of quality measuring tools, these tools do not praise the quality of the program but point out problems, thus it makes them have feelings of being not good at programming. Besides, the quality measurement tools presents the measurement result as it is. Since it does not define the criteria of program quality, students cannot judge how high their program quality is. As a result, it is difficult for students to utilize these tools and it does not lead to improve the program quality. In this paper, we propose an educational method in which visualize the program quality by introducing a concept called *achievement*, which is often used in video games. Application experiment reveals that the proposed method is effective to improve the quality of the student-created program, students awareness for program quality, and motivation for programming.

Keywords: programming education, program quality, automated testing, CI tools, achievement, visualization

1. はじめに

効率的かつ効果的なプログラミング教育の実現を目的と

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a)] k-hanaym@ist.osaka-u.ac.jp
[b)] shinsuke@ist.osaka-u.ac.jp
[c)] higo@ist.osaka-u.ac.jp
[d)] kusumoto@ist.osaka-u.ac.jp

して，自動テストに基づく教育手法[*1]が数多く提案されている [1], [12]．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課題達成の可否を判定する．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，課題達成の可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 [20]，また教員の負担を大きく減らすことができる [4], [10], [34]．

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], [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，またそれを考える意識付けが必要である．

PMD や Checkstyle をはじめとするプログラムの品質計測ツール自体は活発に開発されている．これらのツールはバグの温床となる潜在的な問題の検出や，プログラム品質の客観的な提示が可能である．また，Jenkins に代表される CI ツールとの相性も良く，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，品質計測ツールは問題箇所を事細かに指摘するものであり品質の良さを褒めるものではない [33] ため，プログラミングに対する苦手意識を学生に植え付けかねない．また，それらのツールはメトリクスの計測結果を実測値のまま提示するため，その計測結果には品質の高さに基準が定められていない．したがって，学生に計測結果を提示してもそれをうまく活用することができず，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いえる．

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外的振舞いの正しさだけでなく，内的構造の良さという観点を受講生に意識付けさせることである．この目的を達成するため，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導入し，以下の要件をすべ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のテストを自動で行う．
- 提出されたプログラムの品質計測を自動で行う．
- 学生にも理解しやすい形で品質計測結果を提示する．

従来のテストベースの教育手法と提案手法の概念図を**図 1** に示す．提案手法では，従来のテストベースの教育の流れをふまえたうえで，さらにテストを通過したプログラムに対して品質計測を行う．得られた計測結果を実績達成の可否という表現に変換し，それを学生に提示すること

[*1] Automatic Assessment とも呼ばれる [12].

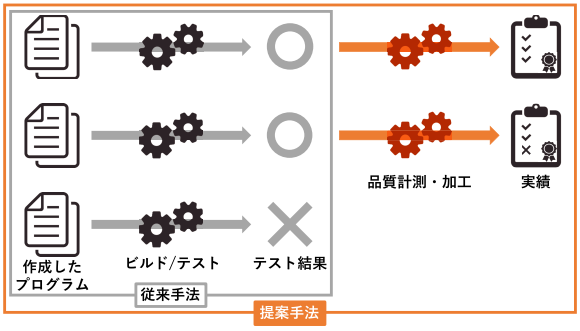


図 1 従来のテストベースの教育手法と提案手法の概念図
Fig. 1 Concept of the conventional and proposed method.

で，プログラムの品質を簡潔に示すことが可能である．また従来の品質計測ツールのように問題のある箇所を羅列し指摘するような形で品質を示すのではなく，問題の**ない**箇所に注目しそれを実績達成という形で褒めることにより，プログラムの品質向上に対する動機付けが可能である．さらに，これらのプロセス（ビルド/テスト，品質計測，実績への加工）はすべて自動で行われる．これにより，迅速な課題達成の可否の判定や教育コストの削減といった従来のテストベースの教育の利点を保持したまま，プログラムの品質を提示できる．

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，大阪大学基礎工学部情報科学科の 3 年生を対象とした授業に導入した．その結果，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を確認できた．加えて，コード行数やサイクロマチック数などの保守性・可読性に直結する品質項目の改善も見られた．

2. 準備

2.1 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，1 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 や Apache Subversion があげられる．

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 ツールの例としては，Jenkins があげられる．

2.2 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の可否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とおりである：

(1) 教員が課題となるプログラムの仕様を策定
(2) 教員がプログラムの振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) 学生が作成したプログラムをテスト
(5) プログラムが全テストを通過すれば課題達成

テストベースの教育では，プログラムの振舞いの正しさを自動かつ即座に評価でき，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 [20]．また，教員は学生の作成したプログラムのすべてに目を通す必要がなくなるため，負担を大きく減らすことができる [4], [10], [34]．

2.3 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様々な利点がある一方，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舞いの正しさのみに焦点を当て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あげられる [7]．ここでの内的構造の良さとは，プログラムの品質モデルとして定められた国際規格である ISO/IEC25010 [*2] の中の一部を意味し，この規格の 4.2.7 項に定められた保守性や可読性に直結する事柄（たとえば「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」など）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うえ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うえ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．

PMD や Checkstyle といったプログラムの品質計測ツールは数多く開発されている．これらのツールはバグの温床となる潜在的な問題の検出やプログラム品質の客観的な提示が可能である．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 ツールを組み合わせることで，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を計測できる．

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 [33]．そのため，計測結果は問題箇所を事細かに指摘した細密かつ膨大なものとなる．たとえば PMD には 300 を超えるルールセットが存在し，「不必要な変数が存在している」といった単純な問題から，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．このような計測結果は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，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．たとえば「`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いえる．

3. 提案手法

3.1 実績の導入

本研究の目的は品質に関する事柄をテストベースの教育に取り入れ，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で品質の高さに基準を設けることができ，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[*3]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以下の 2 種類に大別した：

- **メトリクスベースの実績**：定量的な計測を行う項目に基づく実績
  例：「メソッドあたりの最大コード行数が 30 行以下」「各メソッドのコードカバレッジの平均が 80%以上」
- **ルールベースの実績**：二値的な計測を行う項目に基づく実績
  例：「空の `if` 文が存在しない」「不必要な変数が存在

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

[*3] 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 [35].

しない」

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．たとえば「メソッドあたりの最大コード行数が規定値以下」という実績を考える．この実績を達成することにより，各メソッド内の処理が小さく単純化されるため，可読性・保守性・テストの容易性向上へとつながる．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．

3.3 実績の可視化と比較

提案手法では，実績達成状況の可視化，および他者との比較を行う．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．

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**[*4] 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．

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 [25]（FFE）を用いた．

表 1 使用した外部ツール
Table 1 Used external tools.

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[*5] [25]

[*4] デモページ：https://sdl.ist.osaka-u.ac.jp/~k-hanaym/ave/demo/

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) に示すとおりである．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事前に策定し，Jenkins 上で Gradle と JUnit 5 を用いてプログラムの振舞いをチェックするテストを登録する．
(2) 学生は仕様を満たすプログラムを作成したのち，GitBucket 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 が 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．

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による実績の達成状況を確認する．

4.3 実績の加工方法

品質計測ツールによる計測結果は，以下の手順で実績に加工され，学生に提示される．なお，これら一連の手順は 4.2 節で述べた手順の (8) にあたる．

メトリクスベースの実績

メトリクスベースの実績の場合は，学生の提出したソースコードの品質計測結果が Lv.1 から Lv.3 のどのレベルまで達成しているかを判断する必要がある．まずは品質計測ツールが，ソースコード行数やサイクロマチック数など実

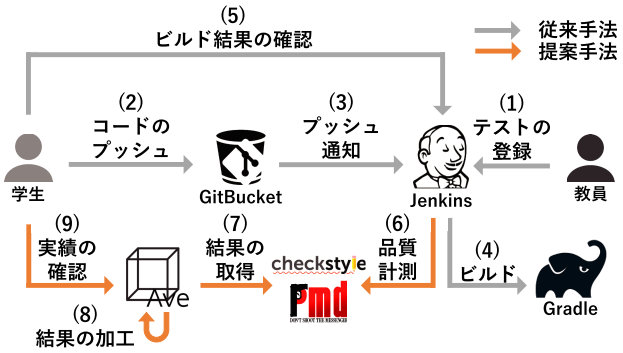


図 2 従来手法と提案手法における授業の流れ
Fig. 2 Class flow in the conventional and proposed method.

[*5] 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．「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できる．

績として定義されている品質項目について計測を行う．次に Ave がその計測結果を取得してパースし，実測値を抽出する．そして，その値が Lv.1 から Lv.3 のどのレベルまでの閾値を上回っている（下回っている）かを判定し，その判定結果を実績として学生に提示する．

ルールベースの実績

ルールベースの実績の場合は，学生の提出したソースコード内に実績として定義されている品質項目の違反箇所がないかを判断する必要がある．ここで実績として定義されている品質項目とは，「空の `if` 文が存在しない」「不必要な変数が存在しない」などを指す．まずは品質計測ツールが各実績における項目について計測を行う．次に Ave がその計測結果を取得してパースし，ソースコード内に違反している箇所がないかを判定する．違反している箇所がまったくない場合は，その実績項目が達成されていると判定し学生に提示する．

4.4 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（**図 3**）
- 実績達成割合の比較画面（**図 4**）
- 品質測定結果の確認画面
- API ドキュメント

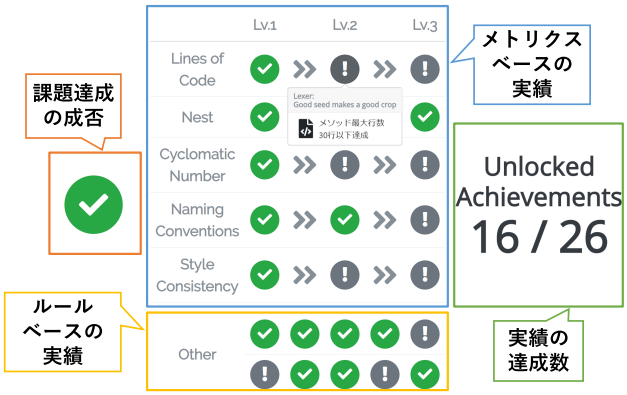


図 3 実績達成状況の確認画面

Fig. 3 Confirmation view of achievement status.

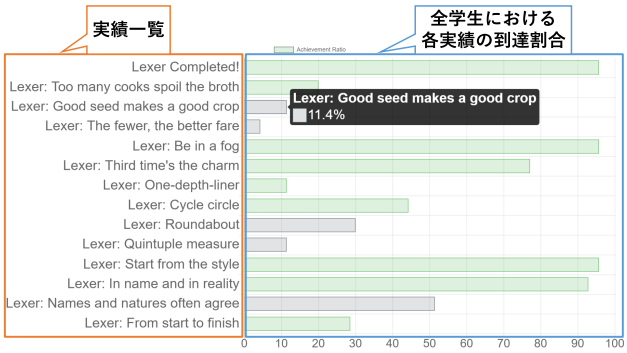


図 4 実績達成割合の比較画面

Fig. 4 Comparison view of achievement ratio.

実績達成状況の確認画面を図 3 に示す．この画面では，課題達成の可否，各実績の概要と達成状況，実績の達成数を確認できる．チェックマーク（“✓”）がついているアイコンは達成済みの実績，エクスクラメーションマーク（“！”）がついているアイコンは未達成の実績を表している．なお以降では，これら各実績を表すアイコンを**実績アイコン**と呼ぶ．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には複数の段階が設けられており，各行が「メソッドあたり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段階のない実績であり複数の難易度が設けられていないため，各行ではなく各実績アイコンがそれぞれの実績項目を表している．すなわち図 3 でルールベースの実績として示されている箇所には 10 個の各実績アイコンがあるため，各々が 1 つの実績項目を表しており，合計で 10 項目の実績に対応している．

各実績アイコンにマウスオーバすると，その実績の概要が表示される．また，実績アイコ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[*6]にジャンプする．この API ドキュメントは Ave の実装と並行して我々が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達成済みの実績，灰色の棒グラフで示されている項目は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．

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．

[*6] https://sdl.ist.osaka-u.ac.jp/~k-hanaym/ave/api/

$Q2$：品質に対する受講生の意識は改善されるか.
$Q3$：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.

またAveを導入することによる副次的な効果として，受講生の作成するプログラムは品質が向上するかも確認を行った（$Qsub$）.

5.2 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（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，演習Dは課題1から課題4の4つの課題で構成されている．本論文では課題1から課題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

Aveで採用したメトリクスベースの実績を**表 2** に，ルールベースの実績を**表 3** に示す．当初は実績項目の候補が多数あげられていたが，最終的に以下の観点を基準として表2および表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
- 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

表2および表3に示した実績項目は，図4の実績確認画面に示した実績項目と対応している．ルールベースの実績については，4.4節で述べたとおりOtherと記された行にある10個の実績アイコンがそれぞれ表3に示す10項目の実績と対応している．また表2におけるCoverageの実績については，一部の課題でのみ使用されている実績であるため，図4中には表示されていない.

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
- 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を用いて解析を行った．また，後半の授業で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．このアンケートには以下のような質問項目を設け，結果として17名から回答を得ることができた.

表 2 メトリクスベースの実績一覧
Table 2 List of metrics-based acihevements.

名前	達成条件	閾値 Lv.1	閾値 Lv.2	閾値 Lv.3
Lines of Code	各メソッドの最大コード行数が規定値以下	50行以下	30行以下	10行以下
Nest	`for` 文，`if` 文，`try` ブロックの各ネストの深さが規定値以下	5以下	3以下	1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	30以下	20以下	10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	30個以下	10個以下	0個
Style Consistency	コーディングスタイルの一貫性が規定値以上	75%以上	90%以上	100%
Coverage	各メソッドのコードカバレッジ（C0カバレッジ）の平均が規定値以上	70%以上	80%以上	90%以上

表 3 ルールベースの実績一覧
Table 3 List of rule-based acihevements.

名前	達成条件	名前	達成条件
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	Declaration	変数，メソッド，クラスなどの適切な宣言を行っている
Switch	`switch` 文を適切に用いている	Import	適切にインポートを行っている
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	Indentation	ソースコードのインデントに一貫性がある
Simplify	簡潔なコードが記述されている	Magic Number	ソースコード内にマジックナンバがない
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	Null	適切に `Null` や `Equals` のチェックを行っている

- Ave 全般や演習 D 全般について記述式で尋ねる項目
  例：「Ave をどれくらいの頻度で使用しましたか？」「Ave を使用してみていかがでしたか？」
- Ave 導入によって自身に表れた効果を 5 段階のリッカート尺度（5 が「そう思う」，1 が「そう思わない」）を用いて評価してもらう項目
  例：「プログラムの品質を改善する重要性が認識できた」「プログラミングに対するモチベーションが向上した」

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．たとえば，字句解析器（Lexer）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にもかか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．

Ave 上位グループと Ave 下位グループに受講生を分けるのは，Ave 使用による効果を確認するため，積極的に Ave を使用している受講生とそうではない受講生を区別するのが狙いである．5.2 節で述べているとおり，演習 D の受講生が Ave を使用するかは任意としている．そのため，受講生全員が Ave を使用してくれるとは限らず，Ave の効果を確認するためには Ave を積極的に使用している受講生を抽出する必要がある．5.5.1 項で述べるとおり，FAC を 10 回以上行った受講生とそうでない受講生で区別することで上位 10% の受講生を抽出できるため，FAC の閾値を 10 回とした．

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と割合を**表 4** に示す．この表から，FAC を 1 回以上行っている受講生と Ave 個人ページを 10 回以上閲覧している受講生は，いずれの課題においても全受講生の半数を超えることが分かる．また，課題 1 と課題 2 に比べて，課題 3 は FAC を行っている受講生が少ないことも分かる．FAC 回数および個人ページ閲覧数で受講生をソートしたところ，受講生 71 名の FAC 回数の総和のうち Ave 上位グループ 6 名による FAC 回数の総和が全課題において 50% 以上を占めており，特に課題 1 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分かる．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分かる．

表 4 FAC 回数と個人ページ閲覧数別の受講生の人数と割合
Table 4 Number and ratio of students by number of FAC and personal-pages view.

内容	10 回以上	1〜9 回	0 回
課題 1 の FAC	9 (12.7%)	42 (59.2%)	20 (28.2%)
課題 2 の FAC	6 (8.5%)	46 (64.8%)	19 (26.8%)
課題 3 の FAC	6 (8.5%)	36 (50.7%)	29 (40.8%)
個人ページ閲覧数	39 (54.9%)	32 (45.0%)	0 (0%)

受講生の実績達成状況について，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おり，この受講生らは全員が 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分かる．

5.5.2 $Q2$：品質に対する意識の改善

受講生の行った FAC の中から，品質に対する意識の改革によるものと見られたコミットを抜粋し紹介する．

変数のファイナライズ

プログラム内で変更が行われていない変数に対し，`final` 修飾子を付加する FAC が存在した．これにより，その変数が不変であることが保証されるためコードの見通しが良くなるほか，再代入によるバグを防ぐことが可能となる．さらに，この FAC ではコミットメッセージに「**ave 用に `final` 定義をした**」と記されていた．Ave の実績の 1 つに「適切に `final` 修飾子を用いている」というものがあり，変更が行われていない変数の宣言時に `final` 修飾子を適切に付加している場合，この実績を達成できる．

コードの簡潔化

処理に `if-then-else` 節を使っていたところを，三項演算子を用いた簡潔なコードに変換する FAC が存在した．不必要な `if-then-else` 節を使わないことで，可読性・保守性が向上する．さらに，この FAC ではコミットメッセージに「**ave の指摘修正**」と記されていた．Ave の実績の 1 つに「簡潔なコードを記述している」というものがあり，不必要な `if-then-else` 節を使わないことで，この実績を達成できる．

マジックナンバの除去

マジックナンバを除去し，`final` 修飾子が付加された定数として宣言する FAC が存在した．これにより，ハードコーディングされていた数値に変数名が付与されるため，意味が分かりやすくなり可読性が向上する．また，数値の変更を一括して行えるようになるために保守性も向上する．Ave の実績の 1 つに「マジックナンバがない」というものがあり，プログラム内にマジックナンバを記述しない

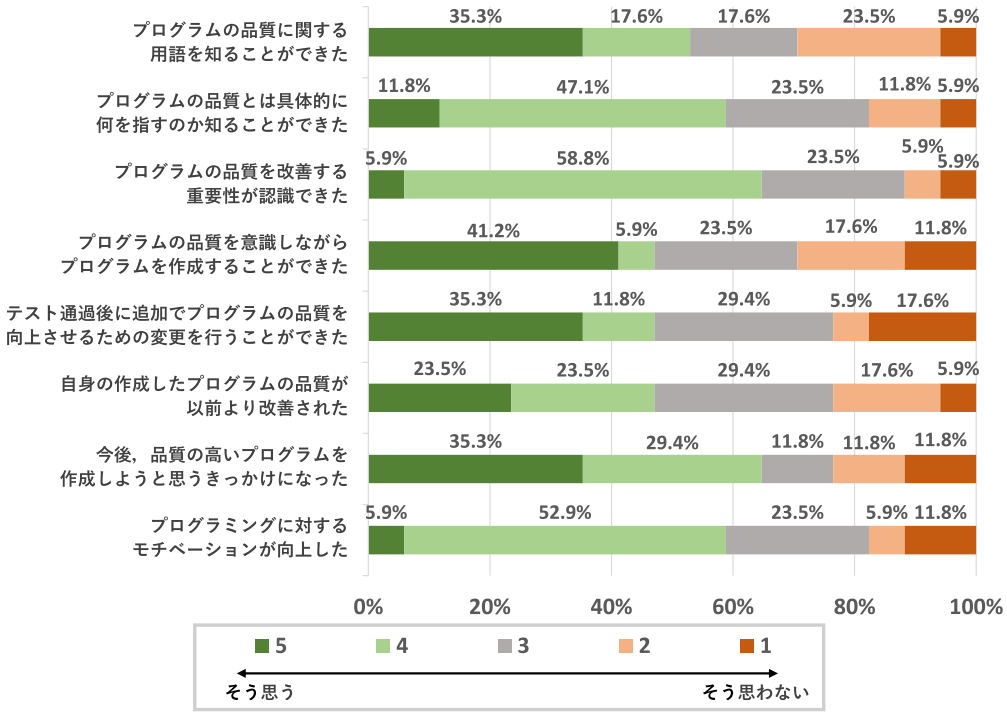


図 5 5 段階評価によるアンケートの結果
Fig. 5 Questionnaire results based on a 5-level evaluation.

ことで，この実績を達成できる．

以上のことから，これらの FAC は Ave 導入により意図的に行われており，実績の達成を目的としていると考えられる．そのほか，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 5** に示す．図に示すとおり，多くの受講生が品質に対する意識の改善を実感しており，特に「プログラムの品質を改善する重要性が認識できた」「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」という質問項目では約 65%の受講生が肯定的な意見（4 または 5）であった．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 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．）

> 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た．（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．）

> できるだけ簡潔になるように意識したり，マジックナンバを除去したりと，分かりやすいコードを書く良い機会になったと思う．（同上）

との記述を行っている受講生もいた．なお，括弧内に各回答が得られた質問文を記している．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革を複数の受講生に行えたことが分かる．

5.5.3 *Q3*：プログラミングに対するモチベーション向上

図 5 の最下段に示すように，「プログラミングに対するモチベーションが向上した」という質問項目に対しては，約 60%の受講生が肯定的な意見であった．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分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った．（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．）

> 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実績一覧をすべてを解除することができた．（同上）

> 今回も Ave の達成を頑張った．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．（同上）

との記述を行っている受講生もいた．なお，括弧内に各回答が得られた質問文を記している．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分かる．

5.5.4 *Qsub*：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を確認した．そこで，Ave 上位グループと Ave 下位グループで作成するプログラムの品質に差があるかを検

表 5 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
Table 5 Quality measurement results of Ave upper and Ave lower group for each task.

内容	課題 1		課題 2		課題 3	
	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

*：有意差があると認められたもの．有意水準 $\alpha = 0.05$.

証した．

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[*7]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 5** に示す．なお表 5 に示す結果は演習 D 終了後に得られたデータを基に算出したものである．

これらの品質にかかわるメトリクスは Ave の実績項目として設定されている．コード行数はプログラムの規模を表す指標として，広く用いられているものである．ソースコードの行数が多いということは規模が大きくそれだけ複雑さも増す傾向にあるため，行数を少なく抑え理解性や再利用性を向上させることが望まれる．次にサイクロマチック数は McCabe によって提案されたメトリクスであり [21]，コード行数と同様に基本的な複雑度メトリクスの 1 つとして古くから知られている．この値が大きいとテストケースを作成するコストがかかるため，メソッド分割などを行って複雑度を低くし保守性を向上させることが好ましいとされている．最後に命名規則違反に関して，一般的にソフトウェア開発やその保守は複数人で行われる．そのため，ソースコード内で自分の記述した箇所だけでなく他人の記述した箇所についても素早く理解できるよう，Java Code Conventions といった開発者が守るべきコーディング規約が存在する [28]．本研究で用いた命名規則違反はその規約の一部であり，規約に従ったコーディングを行うことで可読性や一貫性を向上させることができる．以上に基づき，これらを $Qsub$ を検証するための品質項目として選択した．

表に示すとおり，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，いずれの品質項目においても改善が見られた．また，Wilcoxon の順位和検定（有意水準 $\alpha = 0.05$）を用いて有意差を検定したところ，表に示すとおりほとんどの結果で有意差を確認できた．

6. 考察

$Q1$：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が分かった．また，Ave の使用頻度には受講生間で大きな差があることも確認できた．

課題 1 と課題 2 に比べると，課題 3 において FAC を行っている受講生は少ないことが分かった．これは課題 3 の難易度が他の課題に比べて高く，実績を達成するためにプログラムに変更を加える余裕がなかったためと考えられる．また，実績項目によって達成状況に大きな差が生まれていたが，これは実績項目によって達成するための難易度が異なるためと考えられる．たとえば命名規則違反は，クラス名やメソッド名などを変更するだけで達成できる簡単な実績である一方，コード行数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改善する必要がある比較的難しい実績である．

$Q2$：品質に対する受講生の意識は改善されるか

$Q2$ 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革を複数の受講生に行えたことが分かる．

5.5.2 項で紹介した 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と見られるものがいくつも存在した．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 導入に関する記述式アンケートの回答からも分かるように，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分かる．ただし，どの質問項目においても 1 や 2 といった否定的な意見が存在している．記述式アンケートには，以下のような記述を行っている受講生もいた．

> 課題自体をやることで手一杯だったので，あまり Ave の結果を確認することができなかった．（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．）

> 実績を解除しようとするとかなり難しい項目があったと感じた．（同上）

演習 D は学部 3 年生向けに開講されている授業であり，開発規模はコード行数にして 3,000 行ほどである．また構文解析の理論（字句解析，最左導出，LL/LR 構文解析，意味解析など）も必要なため，授業の難易度そのものが高いといえる．さらに Ave の実績を達成しようとすれば，テス

[*7] 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

ト通過後に追加で変更を加えていく必要があるため，開発にさらなるコストがかかる．このように，演習 D の課題自体や実績項目の難易度が高く Ave を使用する余裕がなかったため，プログラムの品質向上にはつながらなかったと感じた受講生が，Ave に対し否定的な意見を持ったと考えられる．また，図 5 に示すアンケート結果のうち，「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」「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」という質問項目に関しては，他の質問項目より肯定的な意見を持つ受講生が少ない．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 1 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 導入の効果が十分にあったと考えられる．

$Q3$：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

$Q3$ 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分かった．

図 5 の最下部に示すように，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」という質問項目に肯定的な意見であった．また，アンケートに対する記述式の回答からも，楽しい・面白い・分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導入することで，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モチベーション向上につながったと考えられる．否定的な意見を持つ受講生もいるが，この質問項目に否定的な意見を持つ受講生は他の質問項目においてもおおむね否定的な意見を持っていた．すなわち前述したように，Ave を使用する余裕がなかったためにプログラミングに対するモチベーション向上にはつながらなかったと感じている受講生が，Ave に対し否定的な意見を持ったと考えられる．

$Qsub$：受講生の作成するプログラムは品質が向上するか

$Qsub$ 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改善が見られ，Wilcoxon の順位和検定でも有意差を確認できた．これにより，Ave の導入がプログラムの品質向上に貢献した可能性がある．ただし本研究では Ave 上位グループと Ave 下位グループの作成するプログラムを比較および実験しており，Ave 導入の有無による対照実験を行っているわけではない．すなわち，真に Ave 導入が受講生の作成するプログラムの品質向上に貢献しているかは，本研究の結果だけでは判断することができない．加えて，本研究では Ave を積極的に使用しているグループとそうでないグループを区別するために FAC 回数の閾値として 10 回を用いた．FAC 回数はテスト通過後のソースコードの変更回数を示すものであり，Ave の使用頻度を直接計測したものではない．したがって，FAC 回数の閾値を変えることやより直接的な指標を用いることで，Ave 上位と下位グループ間の有意差が得られない可能性がある．

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．また平均命名規則違反数については課題 2 と課題 3 の値が課題 1 の値に比べて増加しており，Ave 上位グループと Ave 下位グループ間の有意差も得られていない．本章「$Q1$：**Ave は実際に使用されるか**」でも述べたとおり，命名規則違反は修正が比較的容易に可能な，簡単な実績である．しかし，課題 2 および課題 3 は課題 1 に比べて難易度が大幅に上昇しており，学生の作成するプログラムもそれにともない規模が大きくなる．そのため修正箇所が増大し，Ave 上位グループ・Ave 下位グループにかかわらずその修正にまで手が回らなかったものと考えられる．

7. 関連研究

7.1 テストベースの教育

テストベースの教育としてはその目的や授業形態などの違いにより様々な手法が提案されている [11], [12], [14], [23], [30], [31], [32]．Edwards ら [8] は学生の提出したプログラムに対し，テストによる自動評価と TA による手動フィードバックの両方を提供できるシステムを提案している．また，Joy ら [15] は提出されたプログラムをオンライン上で自動評価する手法を提案している．既存研究により，テストベースの教育は受講生の成績上昇およびプログラミングに対するモチベーション向上に一定の効果があること [20] や，教員にかかる負担の大幅な削減が可能であること [4], [10], [34] が示されている．

しかし，テストベースの教育はプログラムの品質を軽視する傾向にあることが指摘されている [7], [13]．より具体的には，テストベースの教育では課題達成の可否という二値的な結果のみをフィードバックとして与えることが多い [1]．その結果，剽窃などの不正行為や品質を無視したプログラムの作成を行う傾向が強まる恐れがある [19]．さらに，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 [29]．本研究と手法が類似するものとして，Ohtsuki らによる研究 [26] がある．Ohtsuki らは Jenkins, Git，Checkstyle などのいわゆる DevOps ツール群を用いたプログラミング学習支援システムを提案している．またそのシステムの適用実験では，学生の提出したプログラムの品質向上，迅速なフィードバックの提供，および教員の負担軽減に効果があることが認められている．一方，本研

究では品質計測の結果を実績に加工し提示している．これにより，学生にも理解しやすい形でプログラムの品質を示せるほか，ゲーミフィケーションの要素を導入することでプログラミングに対するモチベーション向上にもつなげることが可能となる．

7.2 ゲーミフィケーションとプログラミング教育

教育に特化したゲーミフィケーションの研究がさかんに行われている [5], [6], [9], [16]．たとえば Khaleel ら [17] は，ゲーミフィケーションをプログラミング教育に導入するための枠組みについて考察している．

さらに，プログラミング文法の習得といった，基礎的なプログラミングの授業に対しゲーミフィケーションを導入した研究も多く存在する [2], [3], [18], [22], [24], [27]．これらの研究では講義資料の閲覧状況や当該講義の理解度を測る簡単な課題に対してゲーミフィケーションを導入し，それらが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を示している．しかし，本研究は文法習得のような基礎的な内容ではなく，コンパイラ開発といった高度な理論に基づく実践・演習形式の授業を想定している．そのため，受講生の作成するプログラムの自由度が高くその品質に差が生まれやすいという特徴があり，プログラムの品質について提示する効果が高いといえる．このような講義において，本研究ではモチベーションや理解度向上といった目的だけでなく，品質に対する受講生の意識改善やプログラムの品質向上を目的として実績を導入し，一定の効果があることを示している．加えて，本研究で提案したシステムは CI ツールと組み合わせて利用可能である．そのため，プログラムの開発，単体テスト，版管理，CI ツールの利用といった，より実践的なソフトウェア開発環境を受講生に体験させることが可能であるという点でも新規性がある．

8. 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．提案手法の適用実験として，大阪大学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，品質に対する受講生の意識改善とプログラミングに対するモチベーション向上，およびプログラムの品質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，Ave の内容面と実装面での柔軟性向上があげられる．内容面の柔軟性向上として，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が考えられる．また，実装面の柔軟性向上として，演習 D に特化していない汎用的なシステムの構築が考えられる．

謝辞 Ave の適用実験に協力いただいた演習 D 受講生の方々に感謝する．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：18H03222）の助成を得て行われた．

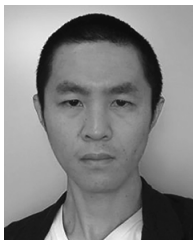
参考文献

[1] Ala-Mutka, K.: A Survey of Automated Assessment Approaches for Programming Assignments, *Computer Science Education*, Vol.15, No.2, pp.83–102 (2005).
[2] Barata, G., Gama, S., Jorge, J. and Goncalves, D.: Engaging engeneering students with gamification, *Proc. International Conference on Games and Virtual Worlds for Serious Applications*, pp.1–8 (2013).
[3] Berkling, K. and Thomas, C.: Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure, *Proc. International Conference on Interactive Collaborative Learning*, pp.525–530 (2013).
[4] Carter, J., English, J., Ala-Mutka, K., Dick, M., Fone, W., Fuller, U. and Sheard, J.: How Shall We Assess This?, *Proc. Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, Vol.35, pp.107–123 (2003).
[5] Denny, P.: The effect of virtual achievements on student engagement, *Proc. Conference on Human Factors in Computing Systems*, pp.763–772 (2013).
[6] Dicheva, D., Dichev, C., Agre, G. and Angelova, G.: Gamification in Education: A Systematic Mapping Study, *Journal of Educational Technology & Society*, Vol.18, No.3, pp.75–88 (2015).
[7] Edwards, S.H.: Rethinking computer science education from a test–first perspective, *Proc. Conference on Object–Oriented Programming, Systems, Languages, and Applications*, pp.148–155 (2003).
[8] Edwards, S.H. and Perez-Quinones, M.: Web-CAT: Automatically grading programming assignments, *Proc. Conference on Innovation and Technology in Computer Science Education*, Vol.40, No.3, pp.328–328 (2008).
[9] Frazer, A., Argles, D. and Wills, G.: The same, but different: The educational affordances of different gaming genres, *Proc. International Conference on Advanced Learning Technologies*, pp.891–893 (2008).
[10] Harris, J.A., Adams, E.S. and Harris, N.L.: Making program grading easier: But not totally automatic, *Journal of Computing Sciences in Colleges*, Vol.20, No.1, pp.248–261 (2004).
[11] Helmick, M.T.: Interface-based programming assignments and automatic grading of java programs, *Proc. Conference on Innovation and Technology in Computer Science Education*, Vol.39, No.3, pp.63–67 (2007).
[12] Ihantola, P., Ahoniemi, T., Karavirta, V. and Seppälä, O.: Review of Recent Systems for Automatic Assessment of Programming Assignments, *Turkish Journal of Gastroenterology*, Vol.18, No.4, pp.265–267 (2007).
[13] Jansen, J., Oprescu, A. and Bruntink, M.: The impact of automated code quality feedback in programming education, *Proc. CEUR Workshop*, Vol.2070, pp.1–19 (2017).
[14] Jimenez-Gonzalez, D., Alvarez, C., Lopez, D., Parcerisa, J., Alonso, J., Perez, C., Tous, R., Barlet, P., Fernandez, M. and Tubella, J.: Work in progress-improving feedback using an automatic assessment tool, *Proc. Frontiers in Education Conference*, pp.S3B–9–T1A–10 (2008).
[15] Joy, M., Griffiths, N. and Boyatt, R.: The BOSS Online

Submission and Assessment System, *Journal on Educational Resources in Computing*, Vol.5, No.3, pp.1–28 (2005).
[16] Juho, H., Jonna, K. and Harri, S.: Does Gamification Work? – A Literature Review of Empirical Studies on Gamification, *Proc. Hawaii International Conference on System Sciences*, pp.3025–3034 (2014).
[17] Khaleel, F.L., Ashaari, N.S., Meriam, T.S., Wook, T. and Ismail, A.: The study of gamification application architecture for programming language course, *Proc. International Conference on Ubiquitous Information Management and Communication*, pp.1–5 (2015).
[18] Knutas, A., Ikonen, J., Nikula, U. and Porras, J.: Increasing collaborative communications in a programming course with gamification, *Proc. International Conference on Computer Systems and Technologies*, pp.370–377 (2014).
[19] Kyrilov, A. and Noelle, D.C.: Binary instant feedback on programming exercises can reduce student engagement and promote cheating, *Proc. Koli Calling Conference on Computing Education Research*, pp.122–126 (2015).
[20] Kyrilov, A. and Noelle, D.C.: 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).
[21] McCabe, T.J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol.SE-2, No.4, pp.308–320 (1976).
[22] Nah, F.F., Zeng, Q. and Telaprolu, V.R.: Gamification of Education: A Review of Literature, *Proc. International Conference on HCI in Business*, Vol.8527, pp.401–409 (2014).
[23] Nobuo, F., Yukiko, M., Toru, N., Kan, W. and Noriki, A.: A Java Programming Learning Assistant System Using Test-Driven Development Method, *International Journal of Computer Science*, Vol.40, No.1, pp.38–46 (2013).
[24] O'Donovan, S., Gain, J. and Marais, P.: A Case Study in the Gamification of a University–level Games Development Course, *Proc. South African Institute for Computer Scientists and Information Technologists Conference*, pp.242–251 (2013).
[25] Ogura, N., Matsumoto, S., Hata, H. and Kusumoto, S.: Bring your own coding style, *Proc. International Conference on Software Analysis, Evolution and Reengineering*, pp.527–531 (2018).
[26] Ohtsuki, M., Ohta, K. and Kakeshita, T.: Software engineer education support system ALECSS utilizing DevOps tools, *Proc. International Conference on Information Integration and Web-based Applications and Services*, pp.209–213 (2016).
[27] Olsson, M., Mozelius, P. and Collin, J.: Visualisation and gamification of e–Learning and programming education, *Electronic Journal of e-Learning*, Vol.13, No.6, pp.441–454 (2015).
[28] Oracle: Java Code Conventions, Oracle (online), available from ⟨https://www.oracle.com/technetwork/java/codeconventions-150003.pdf⟩ (accessed 2019-09-28).
[29] Pettit, R., Homer, J., Gee, R., Mengel, S. and Starbuck, A.: An Empirical Study of Iterative Improvement in Programming Assignments, *Proc. Technical Symposium on Computer Science Education*, pp.410–415 (2015).
[30] Sauvé, J.P. and Abath Neto, O.L.: Teaching software development with ATDD and easyaccept, *Proc. SIGCSE Technical Symposium on Computer Science Education*, Vol.40, No.1, pp.542–546 (2008).
[31] Sauvé, J.P., Abath Neto, O.L. and Cirne, W.: EasyAccept: A Tool to Easily Create, Run and Drive Development with Automated Acceptance Tests, *Proc. International Workshop on Automation of Software Test*, pp.111–117 (2006).
[32] Solomon, A., Santamaria, D. and Lister, R.: Automated Testing of Unix Command-line and Scripting Skills, *Proc. Information Technology Based Higher Education and Training*, pp.120–125 (2006).
[33] Tomas, P., Escalona, M.J. and Mejias, M.: 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).
[34] Tremblay, G. and Labonte, E.: Semi-Automatic Marking of Java Programs Using JUnit, *Proc. International Conference on Education and Information Systems: Technologies and Applications*, pp.42–47 (2003).
[35] 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).

華山 魁生 （学生会員）

2019 年大阪大学基礎工学部情報科学科卒業．同年より同大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程在学中．プログラミング教育およびプログラムの品質に関する研究に従事．



柗本 真佑 （正会員）

2010 年奈良先端科学技術大学院大学博士後期課程修了．同年神戸大学大学院システム情報学研究科特命助教．2016 年大阪大学大学院情報科学研究科助教．博士（工学）．エンピリカルソフトウェア工学の研究に従事．

肥後 芳樹 （正会員）

2002 年大阪大学基礎工学部情報科学科中退．2006 年同大学大学院博士後期課程修了．2007 年同大学院情報科学研究科コンピュータサイエンス専攻助教．2015 年同准教授．博士（情報科学）．ソースコード分析，特にコードクローン分析，リファクタリング支援，ソフトウェアリポジトリマイニングおよび自動プログラム修正に関する研究に従事．JSSST，IEEE 各会員．

楠本 真二 （正会員）

1988 年大阪大学基礎工学部情報工学科卒業．1991 年同大学大学院博士課程中退．同年同大学基礎工学部助手．1996 年同講師．平成 11 年同助教授．2002 年同大学大学院情報科学研究科助教授．2005 年同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価，プロジェクト管理に関する研究に従事．IEICE，JSSST，IEEE，JFPUG，PM 学会，SEA 各会員．