# テスト駆動開発手法によるプログラミング演習

宮崎壮麻[1]・北英彦[*1]・高瀬治彦[*1]
Email: 423M247@m.mie-u.ac.jp

*1：三重大学工学研究学科

◎Key Words　　プログラミング教育，テスト駆動開発，コーディング支援

## 1. はじめに

演習形式で実施されているプログラミング授業では、受講者が課題の内容を正確に理解できないまま進むことや受講者が提出したコードに動作エラーが含まれていることがある。

著者らの研究室では、受講者がコーディングを円滑に進められるように、また、講師やティーチングアシスタント（TA）が適切なタイミングでアドバイスができるようにプログラミング演習システム PROPEL の研究開発および授業での運用を行っている。

PROPEL におけるコーディングの支援は受講者がプログラムを書き始めてからシステムまたは講師や TA がアドバイスを行っている。そのため、課題の内容が理解できない、または、課題の内容が理解できてもプログラミング言語の機能でどう書いたらよいか分からない受講者に対する支援はできていない。

また一方、受講者による動作テストが十分でないため、提出したプログラムに動作エラーが含まれていることがある。これについても、受講者の人数が多く、多数のプログラムをほぼ同時に講師や TA が確認する必要があり、十分な対応ができていない。

そこで、著者らはテストファーストで開発を進めるテスト駆動開発に着目し、受講者のコーディングを支援する機能を提案する。具体的には、以下の機能を提供する。

（1）課題の要求を満たすプログラムに対するテストケースを作成させる。
（2）PROPEL がテストケースの妥当性を評価し、その結果を受講者に伝える。不十分な場合は、不足しているテストケースの数を受講者に伝え見直しをさせる。
（3）従来と同様に PROPEL 内のエディタを用いてプログラムを作成させる。完了したら受講者が作成したすべてのテストケースを使って PROPEL が動作確認をおこない、動作テストの結果を受講者に伝える。動作エラーがある場合は、通過できなかったテストケースがどれであるかを受講者に伝えプログラムの修正をさせる。

これらによりプログラムの作成が始められない受講者でもひとつずつのテストケースを通過するコードを書いていけばよいのでプログラムの作成が始められると期待できる。また、受講者が作成したテストケースが妥当であることは PROPEL が確認しているので十分なテストが行われる。

## 2. 従来の演習方法

三重大学工学部総合工学科電気電子工学コースではプログラミングに関して、1 年次後期に C の、2 年次前期に Java の演習を行っている。

演習環境は図 1 に示す著者らの研究室で研究開発および授業での運用を行っているプログラム演習システム PROPEL である。PROPEL は受講者がコーディングを円滑に進められるように、また、講師や TA は適切なタイミングでアドバイスが行える機能を備えている。

現在、Eclipse や IntelliJ IDEA といった C や Java に対応した一般に普及する統合開発環境（IDE）がいくつもある中で当研究室が独自の IDE を開発する理由としては、以下の通りである。

（1）多人数での演習をサポートできる
（2）使い方がシンプルで初学者でも使いやすい

また、演習の際に使われる PROPEL の機能を受講者が使う機能と講師および TA が使う機能に分けて以下に示す。

（1）受講者が使う機能
① エラーの早期発見機能(1)
システムによる 1 行単位で分かる構文エラーの指摘
② 記述したプログラムの可視化機能(2)
コンパイル済みのプログラムの逐次動作を可視化
③ 対話型 AI を適切に用いるための機能(3)
プログラムのコード全体ではなく処理の手順について対話型 AI を使って学ぶ、また、デバッグの支援を受ける

（2）講師および TA が使う機能
① エラーランキング機能(4)
早期エラー、コンパイルエラー、スタイルエラーをランキング形式で表示し、指導が必要な受講者を早期発見
② 課題の進捗状況の表示機能(4)
座席表の受講者の座席の文字と背景を用いて、どの課題に取り組んでいるかを表示、また、作業が止まっている受講者の背景色を黄色（5 分）や橙色（15 分）で表示

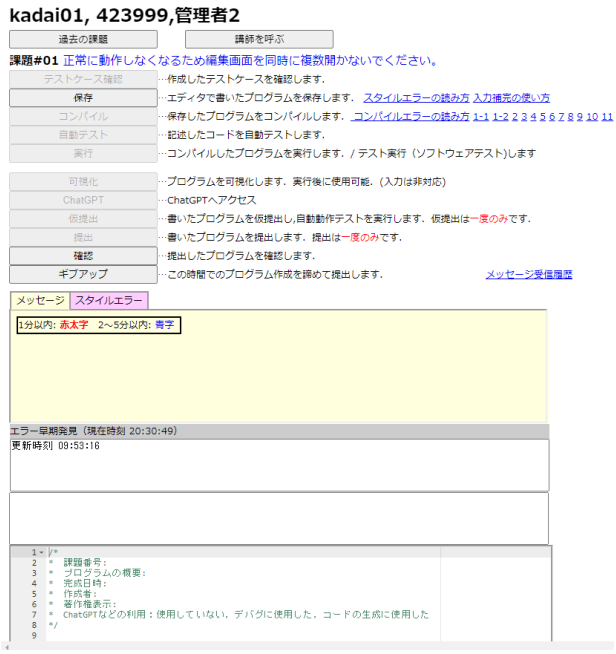


図 1　PROPEL の学習者用画面


図 2　PROPEL の指導者用画面

## 3.　従来の演習の問題点

前述のとおり、著者らはプログラミング演習において PROPEL を利用し、授業を進めている。しかし、PROPEL に備わる機能や支援で解決することができるエラーは主にコンパイルエラーと実行時エラーで、コードを書き始められる受講者に対応したものとなっており、課題の内容が理解できない、または、課題の内容が理解できてもプログラミング言語の機能でどう書いたらよいか分からない受講者に対する支援はできていない。

また、受講者による動作テストが十分でないため、提出されたプログラムに動作エラーが含まれていることがある。これについても、受講者の人数が多く、多数のプログラムをほぼ同時に講師や TA が確認する必要があり、十分な対応ができていない。

## 4.　提案手法

3.で示した問題点に対して本研究ではテスト駆動開発手法を用いた演習を解決手法として提案する。具体的には、課題の内容が把握できない受講者にはコーディングに先んじて行う、テストケースを作成する段階で課題内容の理解を促す。

また、コーディングが始められない受講者に対してはテストケースをそのままの形で実装させることで対応する。このような方法で受講者にコーディングを行わせるのは実際のテスト駆動開発手法に則ったものであり、やむを得ない場合とは意味合いが異なる。また、この時記述されたプログラムは 4.8 または 4.9 に従って修正され、提出時には保守的で可読性の高いプログラムに整形される。

### 4.1　テスト駆動開発

テスト駆動開発とはテストファーストを利用した開発手法である。テストファーストは、プログラムを開発する際に、先行して開発するプログラムの動作を検証するテストコードを先行して記述する。その後、テストを通過できるように本体のプログラムを構成するコードを記述する手法である。

また、テストファーストに加えて記述したコードのリファクタリング（プログラムの動作を保ったまま、実行の効率化と可読性を向上させること）とテスト実行を繰り返しおこなうことで可読性と保守性に優れたコードを記述する手法がテスト駆動開発である。

具体的な開発の基本サイクルとしてはレッド、グリーン、リファクタリングの 3 つであり、以下にそれぞれについての詳細を記述する。

（1）レッド

作成したいシステムの本体のプログラムを記述する前に、テストコードのみが実装されている状態。この時におこなわれるテストは必ず失敗する。

（2）グリーン

先行して記述したテストコードを通過する本体のプログラムが実装されている状態。この時におこなわれるテストは成功する。

（3）リファクタリング

テストを通過した本体プログラムを、その状態を維持しつつ、コードから不必要な部分を取り除き、実行の効率化や可読性の向上を目指す。

テスト駆動開発では、これらの 3 つの状態を 1 つのサイクルとして繰り返し実行し、開発を進める。特に、初めてレッドからグリーン状態に移行するために作成する本体のプログラムは先行して記述したテストを通過することのみが条件である。

### 4.2　テスト駆動開発を演習に採用する理由

まず一般のシステム開発でテスト駆動開発を利用する際の利点と欠点を以下に示す。

（1）利点

① 開発の早い段階で不具合を検知できる
② システムの要件や仕様を理解しやすくなる
③ 開発者にかかる負担が軽減される

（2）欠点

① テストコードの実装や保守に時間がかかる
② 開発手法に慣れるまでに時間がかかる
③ 開発のコストが大きくなる

上記に示す利点のうち、演習に大きく寄与するものは「システムの要件や仕様を理解しやすくなる」と「開発者にかかる負担が軽減される」ことである。

前提として、大学で演習を受ける受講者の多くが

Java をこれまでに扱ったことがない初学者である。そのため、従来の演習だけでは課題で要求されている内容を理解できない受講者や課題の内容が理解できても、与えられた情報を使ってプログラミング言語の機能でどのようなコードを記述するべきか分からない受講者がいる。

そこで、課題を与えられてからすぐにコーディングを行わせるのではなく、テストケースから考えさせる。そのため、この手法では作業を始める前に課題を理解することを受講者に求め、課題を理解した受講者はテストケースの作成に取り掛かる。テストケースの作成を完了させた受講者は、テストケースをもとにコーディングを行う。

また、この時に行われるコーディングは従来の演習のものとは異なり、テストケースを満たすことができればどのような記法でもよい。そのため、コーディングが受講者に与える負担が軽減される。

以上の理由から、本研究ではテスト駆動開発手法を演習形式で実施されているプログラミング授業に導入することを提案する。

また、テスト駆動開発における欠点については演習レベルであればシステム側で十分に対処可能である。特に、実際のテスト駆動開発で大きな欠点となる「テストコードの実装、保守」については研究では受講者にかかる負担を考慮してテストケースのみを作成させ、テストコードの生成はシステム側のみで実行している。そのため、受講者はテストコードの生成や保守を行うことなく、テストケースのみでテスト実行まで可能である。以上の理由から本研究においてテスト駆動開発における先述の欠点は十分に補っているものとする。

### 4.3 テスト駆動開発手法を用いた演習

前述の 3.で提示した課題に対応するためにテスト駆動開発手法を取り入れたプログラミング演習を提案する。具体的な手順としては以下の通りである。

（1）テストケースの作成
（2）テストケースの確認
（3）コーディング
（4）保存とテスト実行
（5）コードの修正（テスト失敗時）
（5）リファクタリング（テスト成功時）
（6）コード提出

以下では、これら各手順について詳細を記述する。

### 4.4 テストケースの作成

テストケース作成用のユーザーインターフェースを組み込んだ PROPEL の学習者用画面を図 3 に示す。テストケースの作成にあたり、受講者はまず課題の内容から何を要求されているのかを読み取り、必要となるテストケースを作成する。

テストケース作成用のユーザーインターフェースには各課題に対応した入力の変数と期待する出力を 1 組のペアが表示されている。

受講者はテストケースを作成するにあたり、入力が予想される変数と期待する出力を考えることで、4.5 でおこなうコーディングに必要となる情報が整理される。

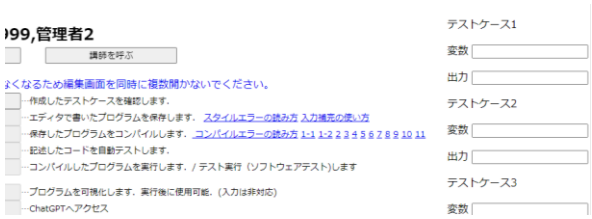


図 3　テストケース作成インタフェースを組み込んだ PROPEL の学習者用画面

### 4.5 テストケースの確認

テストケースの作成を完了した受講者は次に自身が作成したテストケースの妥当性を評価する。しかし、プログラミング初学者である受講者自身がテストケースの評価を行うのは負担になる。そのため、本研究では PROPEL でテストケースの妥当性を評価することで対応する。

テストケースを作成した受講者は図 4 の左上に示されているテストケース確認ボタンを押下することで自身が作成したテストケースの妥当性を評価できる。

現在、システム内部にはテストケースの確認のためにテストケースを全て通過することができるコードを配置することで受講者のテストケースを評価している。

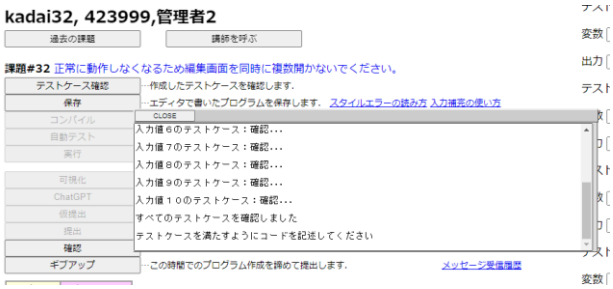


図 4　テストケース確認画面

### 4.6 コーディング

前工程でおこなったテストケースの作成および確認を終えた受講者はコーディング作業を始める。従来の演習では、課題内容の把握をした後、すぐにコーディングに取り掛かる。

一方で、本提案はテストケースを事前に用意してからコーディング作業を始める。そのため、コーディングを始めた時には課題で要求されている内容や、これから受講者自身がどのようなメソッドを書くべきかを把握した状態でこの作業に臨むことができる。

また、コーディング時にはテストケースを満たすことができるという 1 つの条件のみを意識して作業に取り掛かることができるため、記法についての迷いが生じることはなく、コーディングによる心理的負担が軽減されることが期待できる。

### 4.7 保存とテスト実行

4.5 でのコーディング作業を終えた受講者は、コードの保存並びにコンパイルをおこない、テストを実行する。テストの実行では図 5 に示す自動テストボタンを押下することでシステム側が 4.3 および 4.4 で作成されたテストケースをテスト実行に利用できるように加工し、自動テストの準備をおこなう。

準備が完了したら、実行ボタンを押下することで受講

者が記述したコードがテスト実行される。テストを実行する際には、余分な情報から受講者を遠ざけるためにテスト実行のプロセスは学習者用の画面には表示しない。

また、学習者用の画面には図 5 に示す通り、各実行におけるテスト実行の結果のみを表示する。

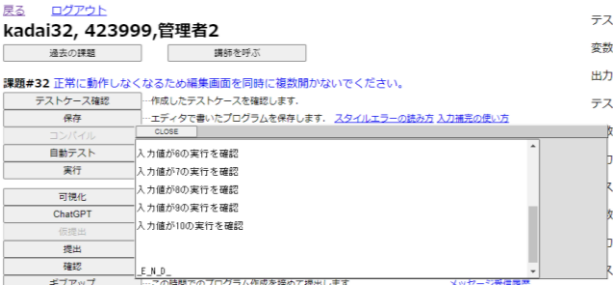

図 5　テスト実行の結果の画面

### 4.8 コードの修正（テスト失敗時）

ここでは 4.6 においてテスト実行に失敗した際の受講者がとる行動を示す。本システムにおいてテスト実行の失敗とは受講者自身が作成したテストケースのうち一つでも実行が確認できない場合を指す。

そのため、テスト実行の結果が表示された際に、確認できていないテストケースがあれば該当するテストケースのみ、実行を確認したことを伝えるメッセージが表示されない。

これにより受講者はどのテストケースが実行できていないかを知ることができるため、速やかに自身が記述したコードにおける修正すべき動作エラーの内容を知ることができる。

### 4.9 リファクタリング（テスト成功時）

ここでは 4.6 においてテスト実行に成功した際の受講者がとる行動を示す。本システムでのテスト実行の成功はすべてのテストケースでの実行が確認できることである。そのため、テスト実行成功時にはコード自体に不備はない。

しかし、テスト駆動開発手法を利用する都合上、1 回目のコーディング作業ではコードの可読性や実行の手順は問われない。つまり、どのようなコーディングをしていてもテストに通過できるレベルであれば 4.9 まで到達できてしまう。

そのため、1 回目にテストを通過したのちにこのようなリファクタリング作業をおこない、コードの可読性の向上と実行手順の効率化を試みた上で 2 回目以降のテスト実行をおこなう。

以上の手順を繰り返しおこなうことで受講者が記述したプログラムを保守的で可読性の高いものにする。

### 4.10 コード提出

4.9 を終え、十分にプログラムのリファクタリングが済んだ受講者は PROPEL 内の提出ボタンからコードの提出をおこない、演習を終える。

## 5.　今後の展開

現在、特定の課題に対応した演習の実施環境は PROPEL に実装済みである。そのため、6 月中旬から 7 月中旬までに現在の受講者を対象とした実験をおこなう。

現在、PROPEL に実装されているテスト駆動開発手法を用いた演習機能は、前期の受講者を対象とした実験をおこなうため、決められた課題にのみ対応できる形で実装されている。

今後は特定の課題だけでなく、利用できる課題を精査したうえで、幅広い演習課題に対応した実装を目指す。

そのため、現時点で実装済みの機能を一般化させる必要がある。加えて、自動テストに必須となるテストコードやテストケースの判定に用いるプログラムの生成といった演習内容ごとに対応が必要とされている問題にも取り組む。

具体的には PROPEL 内でおこなうテストコードやテストケースの判定に用いるプログラムの生成手順の設定だけでなく、今後行う授業で本研究の演習方法を利用できるように講師用の画面についても整える。

## 6.　まとめ

現在、本学科で行われている演習型のプログラミングの授業では課題の内容を正確に把握できない、または、動作テストが行えない受講者がいる。そこで、本研究ではテスト駆動開発手法を用いたプログラミング学習支援の機能を提案した。

本機能は既に PROPEL への実装が完了しているため、今後、プログラミング授業の時間で受講者を対象とした実験を行う。

## 参考文献

（1）小島祐介、高橋功欣、北英彦 : プログラミング演習における効率のよい指導のためのエラー早期指摘、コンピュータ利用教育協議会、PC カンファレンス（2011）

（2）TRAN THANH TUNG、北英彦、高瀬治彦 : プログラムの動作の可視化によるプログラムの動作理解の支援、PC カンファレンス（2022）

（3）井上遙翔、北英彦、高瀬治彦 : プログラミング教育における対話型 AI を用いた学習支援に関する研究、東海支部連合大会（2023）

（4）井冨昌幸、小島祐介、高橋功欣、北英彦 : プログラムの作成状況を把握する機能を持つプログラミング演習システム、PC カンファレンス（2010）

（5）望月将行、森田直樹、高瀬治彦、北英彦、林照峯 : 自動テスト機能を備えたプログラミング演習支援システム、PC カンファレンス（2004）

（6）福本大介、市川嘉裕、山口智浩 : テストケース生成補助に基づくプログラミング学習支援、情報処理学会（2021）

（7）福山裕輝、舩曳信生、中西透、天野憲樹 : テスト駆動開発手法の Java プログラミング教育応用におけるテストコード提出機能、情報処理学会（2009）