プログラミング教育での 生産的失敗を実現する生成 AI 活用の検討

近藤秀樹[1]　遠山紗矢香[2]　井芹俊太郎[1]　石井雅章[1]

概要：プログラミング教育では学ぶべき内容を教師が直接的に教え，学習者がこれを模倣しながら手続きを覚えていく「直接的な説明」と呼ばれる方法が支持されることがあった．短期的に教育目標を達成しやすいものの，概念的な理解に到達することや，学んだことを異なる文脈に応用することが難しいという問題が指摘されている．これに対して「生産的失敗」と呼ばれる教育方法が提案されている．概念的な理解を促進し，学習の転移を引き起こしやすいと考えられているが，プログラミング教育に適用するには困難さがあった．そこで本稿では大規模言語モデルを用いた対話可能な生成 AI を用いてこれらの困難さを解消し，プログラミング教育において生産的失敗が実現できるかを検討する．

キーワード：プログラミング教育，生産的失敗，生成 AI，大規模言語モデル，協調学習

Exploring the Use of Generative AI to Realize Productive Failure in Programming Education

HIDEKI KONDO[†1]　SAYAKA TOHYAMA[†2]
SHUNTARO ISERI[†1]　MASAAKI ISHII[†1]

Abstract: In programming education, a method called "direct instruction" has sometimes been advocated, where teachers directly teach the content to be learned and learners memorize procedures by imitating them. While this approach can easily achieve educational goals in the short term, hence it is difficult for learners to reach a conceptual understanding and apply what they have learned to another context. An educational method called "productive failure" seems to be opposite to the direct instruction. It is expected to promote learners' conceptual understanding and facilitate leaners' transfer of learning, however, there have been difficulties in applying it to programming education. Therefore, this paper examined whether productive failure can be realized in programming education by using a conversational generative AI with a large language model to overcome these difficulties.

Keywords: Programming Education, Productive Failure, Generative AI, Large Language Model, Collaborative Learning

1. はじめに

プログラミング教育の場面では，順を追って手続きを説明する教育方法（以下では「直接的な説明」と呼ぶ）が支持されることがある．直接的な説明とは，学習者にできるようになってほしいことを教師が教え，学習者はそれを模倣しながら手続きを覚えていくことを指す．直接的な説明は，学習者がやり方を自分で見つける発見学習とは区別して説明される場合もある[1]．直接的な説明を受けた学習者は，学んだことをそのままの形で再現することが求められるような試験であれば，よい成績を収める傾向があることが知られている[2]．つまり教師から見れば，試験の方法によっては，短期的に教育目標を達成しやすい教育方法だと言うことができる．

一方で，直接的な説明による授業では，学習者が概念的な理解に到達することが困難なことが少なくないと言われている[3]．学習者は説明を受けた手続きを再現したり，それとよく似た手続きの一部を変更したものを作成したりすることはできるものの，学習した文脈とは異なる文脈に合わせて学んだことを活用できないという，転移の問題が指摘されている[2]．たとえば大学生が入学直後からプログラミングの授業を履修していても，研究室に配属されたときにその力を発揮できるとは限らない．

こうした問題点を内包する直接的な説明に代わる方法として「生産的失敗」という教育方法が提案されている．生産的失敗は，学習者にとって新奇で複雑な問題について，学習者に多様な問題解決の方法を生成したり，より良い解決方法を探し続けたりすることを促す教育方法である[4]．直接的な説明では，教師がまずやり方を説明し，学習者はそのやり方をなぞりながら手を動かすが，生産的失敗では，まず学習者が複雑な問題を解く．その後で教師の説明を受ける場合もあるが[5]，教師の説明がない場合でも転移課題

1 神田外語大学
Kanda University of International Studies, Chiba, Chiba 264-0014, Japan
2 静岡大学
Shizuoka University

の成績は，解きやすく整えられた問題を解いた学習者よりもよい[4]. つまり, 教師が知識や技能を教えるのではなく，学習者が持っている知識や技能を駆使して問題を解決するための試行錯誤を行う．この試行錯誤の過程で学習者は，多様な仮説や自分なりの解法を創り出す．残念ながらこれらの多くは問題を解決できないという意味で「失敗」に終わる．しかし，それら失敗こそが教師の説明を聞く動機づけとなるほか，試行錯誤の過程で学習者が行ったことと教師の説明とが関連づくことにつながると考えられている．

生産的失敗は直接的な説明よりも，学習者の概念的理解の促進を通じて学習の転移を引き起こしやすいと考えられているため，プログラミング教育にも有効だと思われる．しかし，プログラミング教育に生産的失敗の考え方を導入することは以下の理由から容易ではないと考えられる．

(1) 生産的に「失敗」できるようになるまでが長い
プログラミングでは構成要素を一貫した形で組み合わせることで初めてプログラムが動作する．たとえば適切なライブラリを選び，識別子やクラス名，メソッド名，引数を正確に綴り，変数やオブジェクト等を用いる際には予め値を設定したり宣言したりするなど，予め定められたルールを守る必要がある．このため初学者の失敗は，ルール違反による失敗に偏りがちである．一方で「生産的」な失敗は，学習者が解決方針を立てたりプログラムを書いたりする過程で生起するものを指すと考えられる. このため，「生産的」な失敗を初学者が経験できるようになるまでの道のりは長い.

(2) 学習者の既有知識を引き出す問題の設定が難しい
生産的失敗を引き起こすには，学習者の既有知識の活用が必要とされる．学習者が具体的に問題の状況や問題解決の方針を想定しやすい課題ほど，学習者は既有知識を活用しやすくなると考えられる[6]. たとえば普段からパズルゲームを楽しんでいる学習者は，パズルゲームのルールを把握しているため，もし類似したパズルゲームを実装することになった場合にはその経験から得られた知識が活用できるだろう．しかし既有知識や先行経験は学習者個々人で異なっているため，すべての学習者に適した課題を教師が設定することは困難である. また, 学習者に別々の課題を与えた場合，それらに取り組む学習者に教師が一人で対応することは現実的ではない.

(3) 多様な仮説や解決方法を思いつくことが難しい
生産的失敗の方法を効果的なものにするためには，学習者は多様な失敗をすることが期待される．しかしながら一人で取り組む場合は，様々な仮説や解決方法を思いつくことには限界がある．二人以上で話し合うことによって互いに異なる視点から気づきが共有されることで，一人で産出するのは困難なアイディアが出やすくなるため[7]，一人よりも二人の方が好ましいと考えられる．

(1)については，環境構築や文法エラーといった失敗も，生産的失敗の範囲に含まれるという考え方も想定される．しかし本研究では，プログラム生成過程で起こる仮説検証型の活動を重視した場合，環境構築や文法エラーといった失敗は表層的な失敗，学習者がプログラミングを通じて解決したい課題において仮説検証する過程で起こる失敗は生産的失敗と捉える．以上を踏まえると，生産的失敗の考え方をプログラミング教育に持ち込むためには，学習者が自らの既有知識や先行経験などを活用しながら，できるだけ多くの問題解決方法を生成できるような協調学習環境を実現することが有効だと考えられる．

そこで本研究では，大規模言語モデルを用いた対話可能な生成 AI(以下，生成 AI と表記)を活用して上記の問題を解決し，生産的失敗の考え方に基づくプログラミング教育を実現することができるか検討することを目的とする．この学習環境では，学習者たちは自らの既有知識や体験に基づき，第一言語による表現から AI にコードを生成させ，生産的失敗が要求する多様な失敗を実現する．そしてその後に，学習者はプログラミング言語やライブラリなどについての知識を示され，AI の書いたコードを修正・改造しながら学びを深めていく．(1) 学習の初期段階で膨大な専門知識を必要とせず，(2) 既有知識を活かし，(3) 他者と問題解決について協調する学習環境となるだろう．

本稿では，正課の授業における学習者たちと生成 AI の対話ログから有望な事例を抽出し，生成 AI を活用したプログラミング教育への生産的失敗の適用の可能性について予備的な検討を行う．

2. 生産的失敗のための学習環境のデザイン

生産的失敗をプログラミング教育に持ち込むために，どのような授業デザインが可能かを検討し，本研究がどのような学習環境を対象としたのかを述べる．

2.1 生成 AI の役割

生産的失敗のためには，多様な問題解決の方法を生成する必要がある．このために生成 AI の利用を検討する．

2024 年現在，OpenAI や Google，Anthropic 等，複数の企業がテキストチャットに似たインタフェースを備えた生成 AI の商用サービスを提供している. これらのサービスの提供する生成 AI は，人間が用いる自然言語による対話が可能である．人間がキーボードから発話内容を入力すると，それに対して生成 AI が応答を返す．多言語での対応も可能であるだけでなく，話題や知識も豊富であり，あたかも人間の発話内容を理解しているかのような応答を返す.

生成 AI の持つ以下に述べるような特徴から，生産的失

敗に必要な支援を実現できる可能性がある．

(1) 自然言語からコード生成できる

商用の生成 AI は，自然言語での対話を通じて，実行可能なプログラムを生成することができる．曖昧な指示であっても実際に動作するソースコードをさまざまな言語で出力できる．たとえば ChatGPT は「スネークゲームを作って」のような指示から，BSD Games で提供される snake に似たゲームのソースコードを極めて高速に生成できる．明らかに人間のコーディングスピードを凌駕している．そしてこのコードを実行環境にコピーすればそのまま動作することも珍しくない．また，生成されたコードに指示やコメントを追加することで，さらに発展させたコードや異なる案を生成することも可能である．

このため，膨大な前提知識を正確に習得することなく学習者たちの考えを動作するコードに結びつけ，多様な問題解決と失敗を生成できる可能性がある．

(2) 個々の学習者の背景や既有知識を活かせる

学習者は対話を通じて，生成 AI に自身の背景や既有知識を伝えることができる．このため，学習者は教師が決めた単一の課題に取り組むのではなく，学習者が自身の既有知識を活用しやすい課題に取り組むことができる可能性がある．さらに，生成 AI は個々の学習者ごとに，その内容に対応する支援を行うこともできると考えられる．

(3) 協調的な対話に参加できる

人間の学習者同士の議論の結果を受けて，生成 AI はさらに対話を続けることができる．ソースコードを生成するだけでなく，学習者のアイデアに対するコメントや改善案などの応答を生成することもある．これらはそのまま利用できる品質の回答ではないかもしれないが，人間の学習者たちがここから新しいアイデアにつなげることが期待できる．

本研究では，学習者は商用サービスとして提供されている対話形式の生成 AI をいつでもどんな目的にでも活用してよいものとし，同時にコーディング性能の高い生成 AI を「授業用 AI」(後述)という形で学習者に提供した．

2.2 プログラミング環境

生成 AI を初学者のプログラミング学習支援に用いるためには，学習者が生成 AI と連携しやすいプログラミング言語や実行環境を選択する必要がある．この際，プログラミング言語は，学習者の問題解決に寄与するような記述力があり，有力なライブラリが揃っていることも重要である．そこで本研究では，p5.js Web Editor[8]をプログラミング環境とした．学習者はウェブブラウザを用いて，p5.js Web Editor のサイトにアクセスすればいつでも，コードを編集してその場で実行することができる．p5.js ライブラリが組み込まれているので，図形の描画やアニメーションも簡単に実現でき，既存のウェブサービスへのアクセスや膨大な JavaScript ライブラリも利用可能である．p5.js Web Editor 上でコードを実行している例を図 1 に示す．

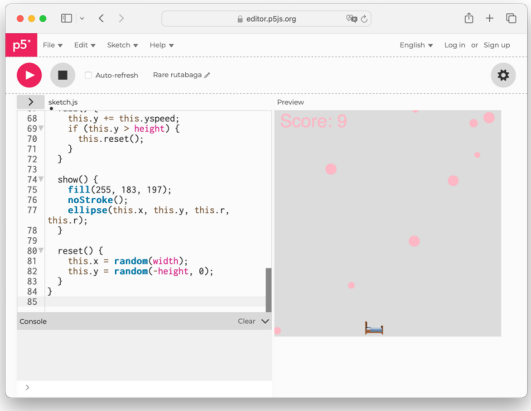

図 1 p5.js Web Editor でコードを編集・実行中の例

図の左側がプログラムを編集する領域である．ここに JavaScript でプログラムを記述する．左上の再生ボタンを押すと，すぐに編集中のコードが実行される．図の左下が JavaScript の標準出力，図の右側が図形の描画領域である．

学習者は，生成 AI の出力したコード全体をコピーし，左側の編集領域にペーストすることで，プログラムを実行することができる．学習者はいつでも，コードを修正したり書き足したりすることができる．

2.3 授業の構成

生成 AI を用いて学習者の多様な問題解決を実行可能なコードに結びつけることができるが，多様でありさえすればよい，ということではない．闇雲に試行錯誤をしても学習者の既有知識に注意が向くことはなく，その後の学習において新たな知識と関連づけられることもない．生産的失敗による学習の効果を高めるような授業を検討する必要がある．また同時に，直接的な説明による授業ではないことを説明することも重要と考えられる．特に，学習者に失敗をさせないように構成されている多くの授業に対して，生産的失敗を導入した授業では失敗が必ず生じるということは，学習者を不安にさせる可能性がある点に配慮が必要である．

本研究では，授業の考え方や進行についての説明を行ったあと，問題解決活動と知識の獲得の順序を入れ替えつつ，自分たちの失敗を振り返りながら繰り返すよう授業を構成した．授業計画を表 1 に示す．1 回 90 分の授業を想定している．

授業の目的は「生成 AI を活用しながら手続き型のプログラミングについての知識や考え方を身につけ，実際に簡

表 1 授業計画

実施日	授業内容
第 1 回	ガイダンス 受講上の注意事項の説明，授業用 AI に関する説明と利用ログの研究利用への同意
第 2 回	生成 AI を用いたソフトウェア開発のチュートリアル 生成 AI の使い方と授業での学び方を理解するワークショップ形式で一通り体験する
第 3 回	第 1 プロジェクト：ペアでのソフトウェア開発(1) 受講生同士で任意のペアを組み，生成 AI を使いながら自分達で企画したソフトウェアを開発する
第 4 回	第 1 プロジェクト：ペアでのソフトウェア開発(2)
第 5 回	第 1 プロジェクト：作品のピアレビュー ペアごとに他のペアの作品 3 点について，評価フォームから評価やコメントをつける
第 6 回	p5.js を学ぶ(1) ウェブ上に公開されている p5.js に関する教材を分担して理解し，互いに説明する
第 7 回	p5.js を学ぶ(2) ウェブ上に公開されている p5.js に関する教材を分担して理解し，互いに説明する
第 8 回	第 1 プロジェクトの改造案をまとめる ピアレビューで受け取った評価やコメント，自分達の反省点を踏まえて改造案をまとめる
第 9 回	第 1 プロジェクトを改造する 改造案を元に実際に作品を改造し，ソースコードの差分を示しながら説明をまとめる
第 10 回	第 2 プロジェクト：ペアでのソフトウェア開発(1) 第 1 プロジェクトとは異なるペアを組み，生成 AI を使いつつ自分達で企画したソフトウェアを開発する
第 11 回	第 2 プロジェクト：ペアでのソフトウェア開発(2)
第 12 回	第 2 プロジェクト：ペアでのソフトウェア開発(3)
第 13 回	第 2 プロジェクト：ペアでのソフトウェア開発(4)
第 14 回	第 2 プロジェクト：ペアでのソフトウェア開発(5)
第 15 回	最終プレゼンテーション

単なアプリケーションのプログラミングを通して，仮説検証や問題解決に関する態度や考え方を養うこと」とした．生成 AI の利用方法や p5.js Web Editor の操作方法などの最低限の知識だけは最初に導入するものの，授業の目的につながるソフトウェア開発に関する知識は説明しない状態でまず開発のための時間をとり，その後にプログラミング言語について学ぶという構成になっている．生産的失敗で主張されている原則を適用したものである．

開発プロジェクトには学習者同士がペアで協調的に取り組むこととした．これは多様な解法を生成することの支援と，複雑な課題に取り組むことの支援に相当する．

第 1 回の授業では次のようなことを説明する．

(1) 授業の進め方と生産的失敗の考え方
生産的失敗と方向性が類似した資料[9]を示しつつ，知識を得てから制作に取りかかるのではなく，まず制作に取り組んでから知識を得るという順序であることを周知する．同時に，自分達で相談しながら建設的に試行錯誤する時間が必要であることを示す．

(2) 授業の目的は理解であり作品ではないことの確認
授業の目的を説明し，よい作品を残すことが目的ではないことを強調する．作品は理解の結果であって，理解深化を促すためにペアで協調することが有効であることを示す．

第 2 回では，それ以降の授業でのプロジェクトへの取り組み方のチュートリアルを実施する．標準的な授業とは異なる構成であることを説明するだけでは学習者にとって不十分である可能性は高い．そのため，学習者がその後の授業で取り組む活動を一通り体験できるようにする．ここで学習者は，生成 AI を利用した制作にかかる時間を実感することができる．また，通常の授業であれば予習や復習に使う時間を自分たちなりの試行錯誤や制作に充てることが，この授業の効果を高めることを伝える．

第 3 回から第 5 回で「第一プロジェクト」を実践し，相互にピアレビューを実施する．このプロジェクトでは，学習者は互いに任意の学習者とペアを組んでソフトウェア開発にあたる．プロジェクトのゴールは「実際に動作する面白い・特徴的なアプリを制作し，それを他の人に試してもらえるようにすること」として提示する．学習者の背景や

既有知識を尊重するため，制作するアプリを例示しない．代わりに生成 AI は自由に利用できるようにすることで，教員が直接支援する必要性を低減する．

その後，第 6 回と第 7 回で学習者は JavaScript に関する教材[10]の内容を学び，その内容をプロジェクトで制作した作品の改造に用いる．ここでは，第一プロジェクトで自分たちが生成したコードの読解と改造のために学ぶことを目標とする．ただし，学習者は教員の講義を受けるのではなく，学習者間で内容を分担して相互に教授した上で，目標達成に必要なことを自主的に学ぶ．そのうえで第 8 回と 9 回では作品の改造を行う．

第 10 回以降では，第一プロジェクトの作品改造経験を踏まえて第二プロジェクトに取り組む．同様の過程を二度経験することで，学習者が学びを深められるよう構成した．

3. 授業用 AI

本研究では，OpenAI の ChatGPT API を用いた対話システム「授業用 AI」を実装して生産的失敗のための学習環境を学習者に提供する．授業用 AI は，チャットアプリケーションのような体験を学習者に提供する．生成 AI としての処理は行わず，実際には学習者からの入力をそのまま生成 AI に引き渡し，生成 AI からの応答を加工せずにそのまま学習者に向けて送信するものである．

既存の生成 AI をそのまま利用せずに授業用 AI のようなインタフェースを実装する理由は以下の 2 点である．

(1) 学習者と AI の対話を記録・分析する

学習者が AI との対話を通じて学習を進める場面では，対話の内容も評価分析の対象とする必要がある．AI が学習者と生成 AI の間に入ることで，学習者の入力したプロンプトや生成 AI の応答を漏れなくテキストデータとして記録できるようになる．

既存の AI サービスでは学習者と生成 AI との対話を機械処理可能な形で網羅的に記録することは難しい．学習者からスクリーンショットなどの形で個別に提供してもらうことはできるが，分析に必要な記録が提供されるとは限らない．また学習者にとってみれば，煩雑な操作をしてまでデータを提供する意義が感じられないだろう．学習過程の分析のためには，自動的で精度が高く，機械処理可能な形で網羅的な記録を取る必要がある．

(2) 一定以上の AI の対話能力を提供する

既存の AI サービスの対話性能はモデルの規模に依存する．学習者の言葉を理解し，かつ，十分な品質のコーディングが可能なモデルは多くの場合，有料で提供されている．一方で学習者が生成 AI を活用する場合，無料のモデルに留まることが予想される．学習環境の一部としてコードの生成に十分な性能のモデルを提供するためには，商用サービスを活用する必要がある．

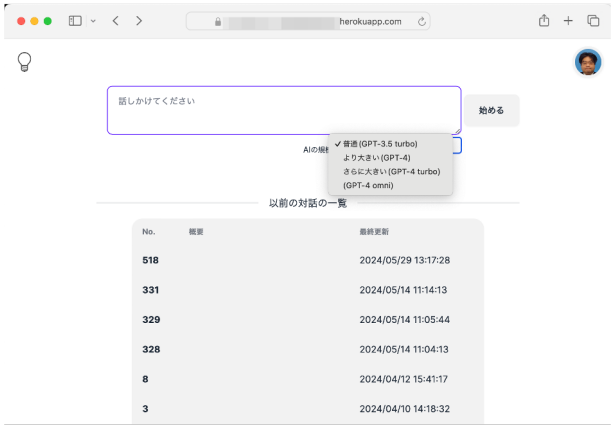

図 2 授業用 AI の初期画面

授業用 AI は Ruby on Rails で実装されており，クラウドサービスのひとつである Heroku 上で常に稼働している．学習者は授業時間中だけでなく，いつでもアクセスして利用できる．学習者はウェブブラウザでこれにアクセスし，Google アカウントで認証した後，生成 AI と対話することができる．

図 2 は授業用 AI にログインし，対話を始める前の初期画面である．テキスト入力欄に最初の発話内容を入力して，「始める」ボタンを押すことで対話セッションが始まる．対話セッションとは，初期画面から対話を始めてからその対話が終了するまでの一連の発話全体のことを指す．対話セッション開始にあたって，どの規模の生成 AI と対話するのかを選択できる．デフォルトでは GPT-3.5 turbo が選択されているが，GPT-4，GPT-4 turbo が任意に利用可能である．GPT-4 omni もリリース翌日から暫定対応として追加された．ただしマルチモーダル機能は利用できず，他のモデルと同様，テキストベースやりとりだけが可能である．

図 3 に対話画面の例を示す．左側に生成 AI を示すアイコン，右側に学習者のアイコンが表示されており，吹き出しの形で発話が表示されている．学習者は下部のテキスト

図 3 授業用 AI での対話例

入力欄に発話を入力し，送信ボタンを押すことで発話を授業用 AI に送信する．授業用 AI はその発話内容を記録し，生成 AI に送信する．ファイル添付には対応していない．

授業用 AI は生成 AI からの応答を Markdown 形式とみなしてレンダリングする．応答の中に含まれるコード部分は明確に他と区別される．図中の黒い背景の部分がコードとしてレンダリングされた部分である．授業用 AI によって，この領域にはコード全体をそのままコピーするボタンが付与される．学習者は吹き出し内の Copy の部分をクリックするだけでコードを正確に応答から持ち出すことができる．そのコードはそのまま p5.js Web Editor にペーストして動作させることができる．

4. 実践

神田外語大学 外国語学部の学生を対象として 2024 年度前期に開講される「ソフトウェアデザイン I B」の授業として実践を行った．この授業は選択必修であり，1 年生から 4 年生までの学生が受講できる．週に 1 コマ 90 分である．教員 1 名とスチューデントアシスタント 1 名で実施した．

履修した学生は 35 名である．カリキュラム上，ソフトウェアの開発を扱う授業はほとんどなく，受講生はプログラミングについての経験が極めて少ないと考えられる．

授業の構成上，ほとんどの時間をペアでの協調活動に充てるため，グループワークのための教室で実施した．受講生は必携化となっているノート型 PC か iPad を持ち込んで受講する．授業の様子を図 4 に示す．授業用 AI の利用は強制ではなく，学習者がすでに活用している生成 AI がある場合はそれを利用してもよいこととした．

5. 期待される結果

6.1 節と 6.2 節では，受講者の活動について概要を示す．6.1 節では，2024 年 4 月 10 日（第 1 回授業日）から 5 月 8 日（第 5 回授業日で，第 1 プロジェクトのピアレビュー）の直前までの実践を対象として，授業用 AI のログから授業用 AI の利用状況を分析した．

6.2 節では，2024 年 7 月 18 日の最終授業で取得したアンケート結果の一部回答を示した．ここでは「授業を受ける前に，プログラミングやソフトウェアのデザインについて抵抗感はありましたか？」および「現在，プログラミングやソフトウェアのデザインについて抵抗感はありますか？」という質問に対する 5 段階評価での回答を分析した．生成 AI を使った生産的失敗のプログラミング教育実践は，学習者がプログラムを読解したりプログラムの内容を理解したりしながらプロダクトを作るうえでハードルが高いと考えられるが，実際に学習者はハードルが高いと感じていたのかを検討するために行った．

図 4 授業中の様子

6.3 節では，授業用生成 AI との対話事例を示した．生産的失敗の実践デザインが機能した場合，学習者は生成 AI との対話の中で少しずつ異なる多様なコードを生成させるようになると考えられる．この際に生成されるコードは，学習者が思い描いているものそのものではないこともある．この意味で「失敗」である．学習者は生成 AI の出力を目の当たりにして何が失敗だったのかを意識することになり，それを踏まえた次のコードを生成させたり，自らコードを修正したりする．必然的に発話回数は多くなり，発話は具体的な内容を含んで長くなり，対話全体も長くなるだろう．さらに学習者は，これら多様な表象の違いを比較検討して，プログラミングについての理解を深めるだろう．対照的に，生産的失敗が機能しない場合，学習者は「失敗」に基づく多様なコードを生成しようとはせず，自分が主観的に納得できるコードを獲得できるまで生成 AI に再生成を繰り返し要求することになるだろう．それでいて多様なコードを求めているわけではないため，発話数は少なくなり，再生成を要求するだけなので発話自体も簡潔で短く，対話も短く終了すると考えられる．

6. 結果と考察

6.1 授業用 AI の利用状況

(1) 対話セッション

学習者と生成 AI との対話セッションは 228 件であった．そのうち，ソースコードを含まない等，授業と関係がないと考えられる対話は 67 件であった(第一著者の判断による)．もっとも多く対話セッションを実施した学習者は分析対象期間を通じて 33 回の対話セッションを実施していた．

(2) 生成 AI の選択

GPT-3.5 turbo の利用が 84 件，GPT-4 は 63 件，GPT-4 turbo は 81 件であった．GPT-3.5 turbo 以外は明示

的に操作しなければ利用できないにも関わらず一定程度以上利用されている．このことから，学習者は生成 AI を意識的に選択していたことが示唆される．

(3) 授業用 AI へのアクセス数

1 度も授業用 AI にアクセスしなかったのは 7 名，1 度だけ授業用 AI にアクセスしたのは 4 名，2 回以上授業用 AI にアクセスしたのは 24 名であった．アクセス回数が 1 回以下の学習者が見られた．

授業時間内の机間巡視において，授業用 AI を利用せず，各自で商用 AI サービスを利用している事例が複数見られた．これらは上記結果に含んでいない．

(4) 発話数

分析対象期間中の学習者と生成 AI の発話数の合計は 3156 件であった．セッション毎の平均発話数は 13.8 である．もっとも多くの対話を行った学習者の発話数は 58 件であった．

6.2 アンケート結果

Google フォームで行ったアンケートについて，履修学生 35 名のうち，29 名から回答が得られた．結果を図 5 に示す．授業前に半数程度の受講生が 4 以上の抵抗感を持っていたが，授業後には 8 割程度の受講生が 2 以下の抵抗感へと変化したと言える．ただし，本結果を解釈するうえで，授業前の主観を授業後に想起させたことの影響があることは考慮する必要がある．

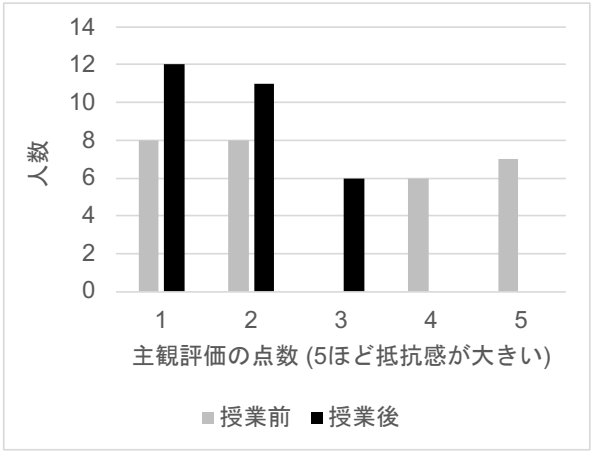

図 5 受講生のプログラミングやソフトウェアデザインに対する抵抗感のアンケート結果

6.3 対話事例

分析対象期間中の全 228 セッションの中から，生産的失敗の成功的な対話を第一著者が選定し，学習者の発話だけ

表 2 生産的失敗が成功したと考えられる発話例

No	発話内容 (人間の学習者がキーボードからタイプした内容)
1	(ソースコードを貼り付け)
3	このプログラムの編集を手伝ってくれませんか？
5	現在の設定では、一行以上揃った場合も一行しか消えなくなっています。二行以上揃った場合もその分消えるように変更できますか？
7	すべてのプログラミングで書いてください。
9	二行以上消えないですケど、何か間違ってないですか？
11	コード全体を送ってもらえませんか？
13	このゲームをベースにアレンジを加えてくださいませんか？
15	途中で障害物が落ちてくるのはどうですか？
17	他になにか面白いアイデアはありますか？
19	テトリスは、四角の背景が一般的ですが、それを円形にすることはできますか？残りの設定は変えなくていいです。
21	関数を省略しないで表示してくれませんか
23	(ソースコードを貼り付け)
25	このプログラミングをアレンジするのを手伝ってくれませんか？
27	(ソースコードを貼り付け)
29	爆弾のような障害が落ちてくるのを追加できますか？ その爆弾を列揃えて消すことができたら 100 スコア追加、60 秒時間も追加され、画面にあるブロックをすべて消せるようにできますか？
31	爆弾が落ちてきません
33	コード全体でくれませんか？
35	大丈夫ですか？

表 3 生産的失敗が成功しなかったと考えられる発話例

No	発話内容 (人間の学習者がキーボードからタイプした内容)
1	p5.us で作動する福笑いのゲームのコードを書いてください。html は使わないでください。
3	作動しません。p5 を使わないでください。
5	作動しません。p5 を使わないでください。
7	作動しません。書き直してください。
9	Turtle がエラーと出ます。書き直してください。
11	これを p5js に書き直してください。
13	SyntaxError: Unexpected identifier 'turtle'. import call expects exactly one argument.というエラーが出ました。p5js のままで書き直してください。

抜粋したものを表 2 に示す. 学習者が制作していたのはテトリスである．学習者と生成 AI で合計発話数 36 件に及ぶ対話セッションである. No はセッション内での発話順を意味する．生成 AI との対話は常に人間から開始され，人間の発話に対して生成 AI は一つの応答を返すため，奇数が人間の発話，偶数が生成 AI の発話となる．生成 AI の応答は長いため省略する．またソースコードは長大なため，ここでは単に「ソースコードを貼り付け」として省略している．

この事例からは，障害物の導入(No.15)や背景の変更(No.19)，爆弾の導入(No.29)など，新しいアイデアを具体的に指示していることや，セッション内で少しずつ継続的に変更を進めていることが読み取れる．これらは生産的失敗で求められる多様な問題解決や失敗と考えられる．

これに対して，生産的失敗の不成功的な対話の例を同様に表 3 に示す. 合計発話数 14 件の対話セッションである．

この事例では，学習者は単に動作しないことを何度も繰り返すだけであり(No.3, No.5, No.7, No.9)，最後にエラーメッセージを伝えることはするものの(No.13)，試行錯誤自体が成立していないことが分かる．発話内容から，p5.js Web Editor 向けのコード生成を指示すれば改善の見込みはあったものの，実行環境について学習者の理解が十分でなかったことが示唆される．

7. おわりに

プログラミング教育において学習者の理解を深めるために，生産的失敗の導入について検討した．多様な問題解決や失敗を実現するために，生成 AI と学習者の対話を学習環境に組み込み，それに合わせた授業案を提案した．生産的失敗の考え方に基づくことや，ペアを組んで協調的に課題に取り組むこと，全員が同じ課題に取り組むのではなくペアごとに自分達で決めた課題に取り組むこと，作品の高度さでなく理解を重視すること(作品の高度さは理解の結果であること)，などの原則を組み込んだ授業案である．文系大学生を対象として授業案を実践し，学期途中までの利用ログをもとに予備的な評価を行った．その結果，初学者に近い学習者であっても，試行錯誤しながら自分なりに多様な問題解決や失敗を実現できる可能性が示された．同時に，単純に生成 AI を導入するだけでは試行錯誤に失敗する事例も見られた．学習者に対する積極的な支援の必要性が示唆された．生成 AI 側のシステムプロンプトなどでの介入も有望な可能性がある．

本稿で検討した活動がその後のプログラミングの学習にどのように活かされるのかは本稿では扱っていない．生産的失敗の考え方に照らせば，ここまでの試行錯誤をもとに，学習者はプログラミングの諸概念を獲得していくはずであるし，概念的理解や転移につながるはずである．今後の授業実践や制作された作品の評価を通して明らかにする必要がある．

参考文献

[1] Klahr, D.: “To every thing there is a season, and a time to every purpose under the heavens”: What about direct instruction? In Constructivist Instruction, Routledge, pp. 291–310 (2009).

[2] Bransford, J. D., Brown, A. L., Cocking, R. R.: How People Learn: Brain, Mind, Experience, and School: Expanded Edition. National Academy Press (2000).

[3] Kirschner, P. A., Sweller, J., & Clark, R. E.: Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational Psychologist, Vol.41, No.2, pp.75–86 (2006).

[4] Kapur, M.: Productive failure. Cognition and Instruction, Vol.26, No.3, pp.379–424 (2008).

[5] Kapur, M., & Bielaczyc, K.: Designing for Productive Failure, Journal of the Learning Sciences, Vol.21, No.1, pp.45–83 (2012).

[6] Lave, J. Cognition in Practice: Mind, mathematics and culture in everyday life. Cambridge University Press (1988).

[7] Miyake, N. Constructive interaction and the iterative process of understanding. Cognitive Science, Vol.10, No.2, pp.151–177 (1986).

[8] “p5.js Web editor”, https://editor.p5js.org/, (参照 2024-06-14).

[9] “「とりあえずやってみる」はなぜ効果的か”. https://note.com/playfulquest/n/n0819da49097f, (参照 2024-06-14).

[10] “文系大学生のための p5.js 入門”, https://zenn.dev/ojk/books/intro-to-p5js, (参照 2024-06-14).