

研究内容&テーマ紹介

榎原

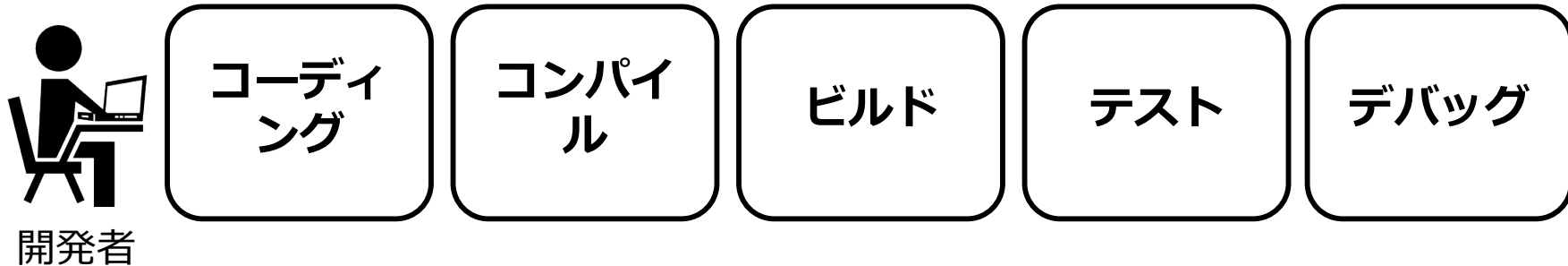
研究背景の紹介：ソフトウェア工学

- **ソフトウェア工学**とは、コンピュータソフトウェアを対象として、その開発、運用、保守における生産性と品質の向上を実現するための技術体系や学問体系*を指す
 - **The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.** --- *IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610.12-1990, IEEE, New York, 1990*
- 近年、ソフトウェアの需要が高まり、実践的な技術を持つソフトウェア開発技術者の育成が急務となっている
 - Industry 4.0: IoT、AI
 - クラウドコンピューティング、ブロックチェーン etc...

背景：ソフトウェア工学教育[IT人材白書2017]

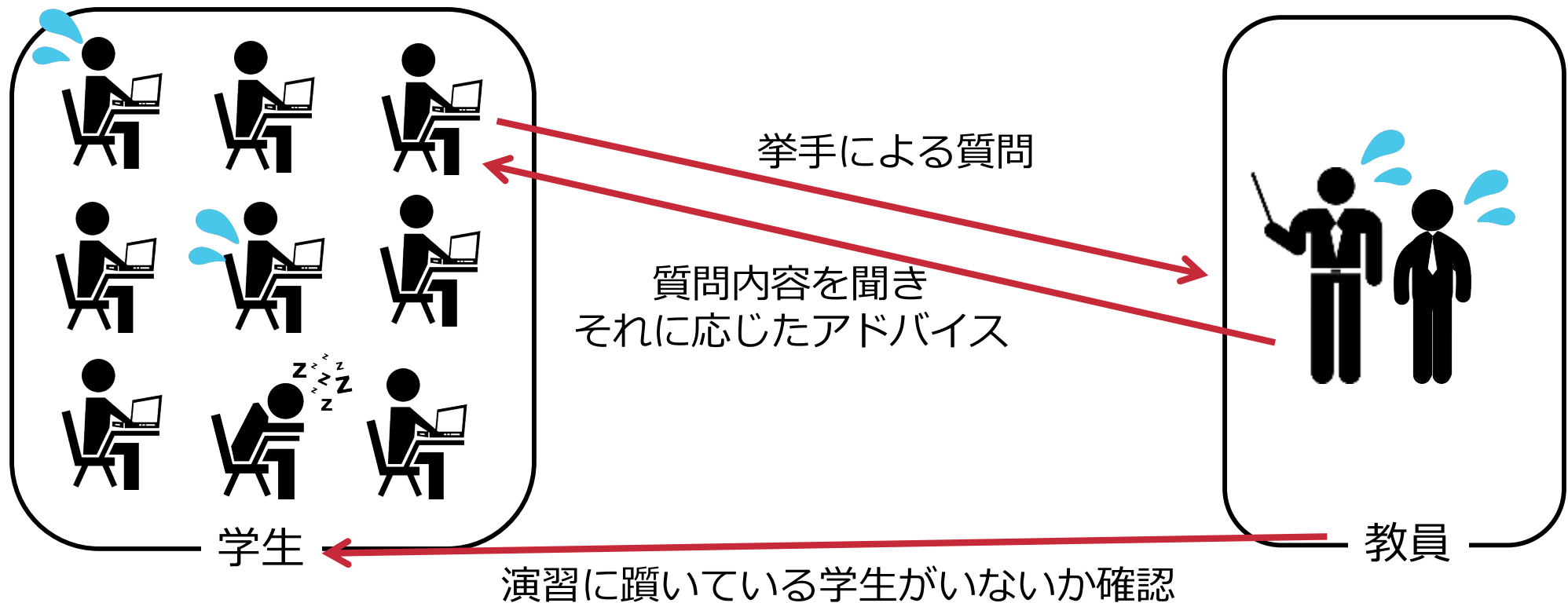
- 近年、ソフトウェアの需要が高まり、情報系学科を有する教育機関において高度IT人材の育成が望まれている
 - 高度IT人材：ITに関する深い知識と創造性を担う人材
- 上記教育機関において、学生が実際にプログラミングを行い実践力を高める形式の科目を開講している割合が増えている
 - プログラミング演習：個人でシステム等を開発する
 - ソフトウェア開発演習：グループでシステム等を開発する
 - Project-Based Learning(PBL)演習：プロジェクトベースでシステム等を開発する
- 上記演習において、プログラミングについて初めて学ぶ学習者(以降初学者と呼ぶ)の教育支援が行われている
 - 受講生の約3割が演習を非修了する[Bennedsen et al. 2007]
 - 教員による適切なサポートが望まれる

背景：プログラミング行動の分析



- 開発者がソフトウェア開発を行うにあたっての振る舞いの内容や成果物、及び関連するメトリクスを**プログラミング行動**と呼ぶ[Ihantola et al. 2016]
- プログラミング行動を収集・分析することで、学生のソフトウェア開発に対する理解度の計測、評価、学習成果に対する暗黙の要因を計ることができる
 - Educational Data Mining(EDM)分野として確立

従来のプログラミング演習の問題点

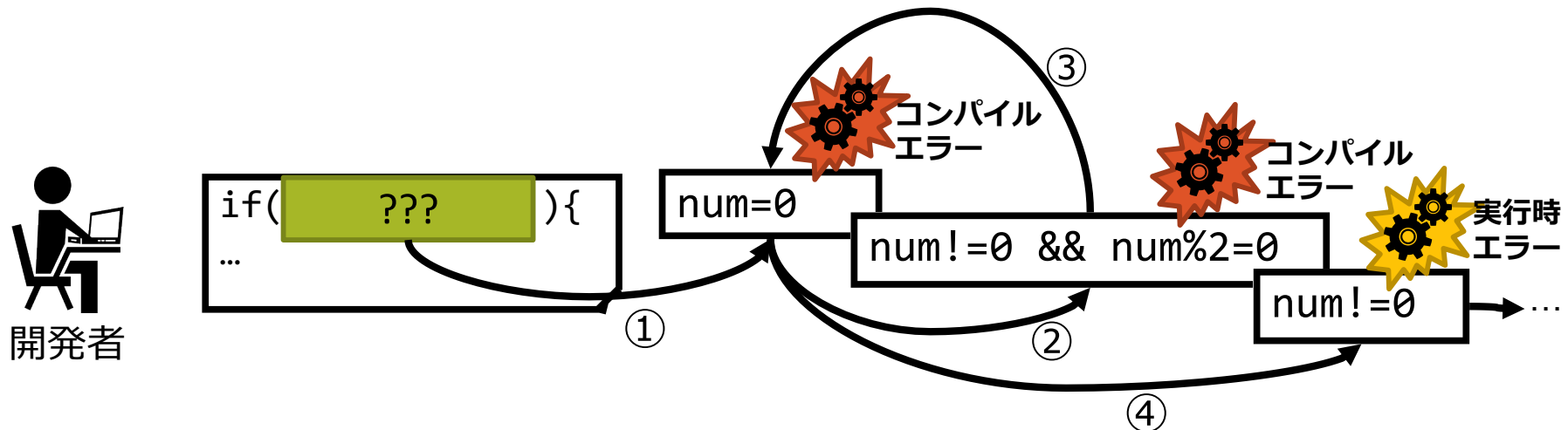


手を上げた学生へ優先的に指導

学生がどのような試行錯誤を行っていたのかが不明確

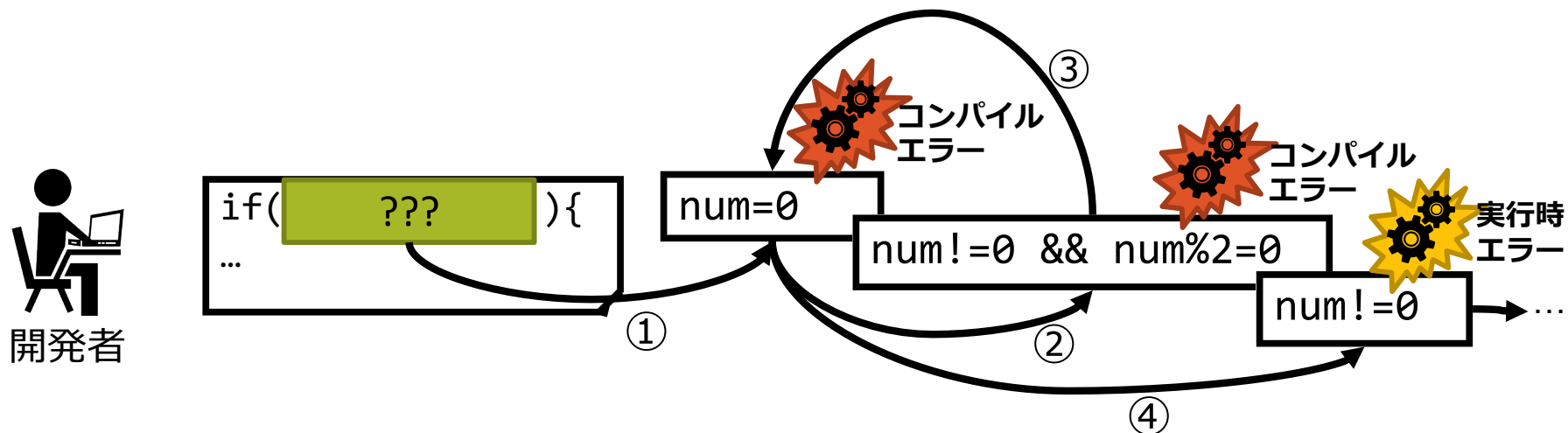
キーアイディア：探索的プログラミング

- プログラミング行動における一連の過程、あるいは特定の段階において、エラーや異常出力を修正するため、あるいは未完成のソフトウェアを完成させるために開発者が行った試行錯誤に関わる振る舞い

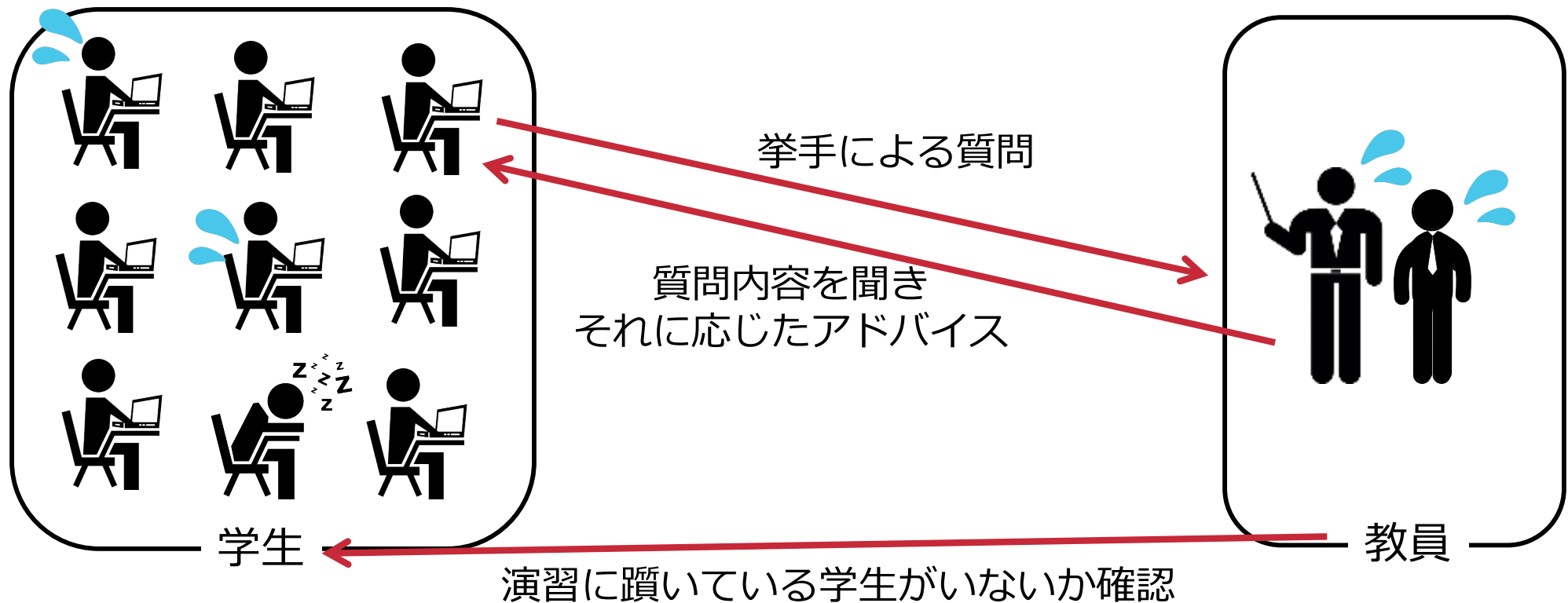


キーアイディア：探索的プログラミング

探索的プログラミングが行われている
= **開発者はプログラミングに躓いている**と仮定
探索的プログラミングを自動検出することで、プログラミングに
躓いている学生を発見、教員が自主的に支援に向かう



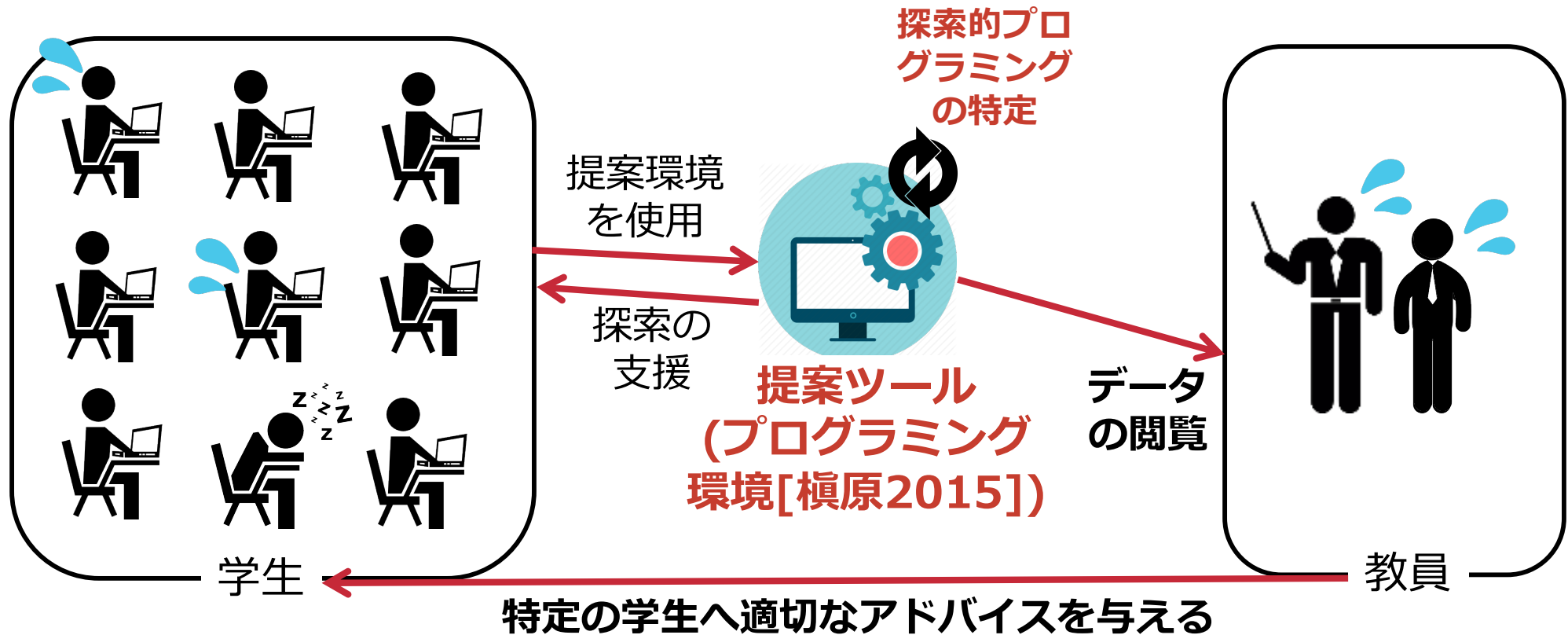
従来のプログラミング演習の問題点



手を上げた学生へ優先的に指導

学生がどのような試行錯誤を行っていたのかが不明確

提案するプログラミング演習支援環境



探索的プログラミングを自動で特定し教員に提示することで
教員が主体的に学生の躓きについて把握、アドバイスが可能

提案ツール

The screenshot displays the C3PV tool interface, which is divided into several functional areas:

- C3PVオリジナル領域 (C3PV Original Area):** The top-left section, highlighted in green, contains the code editor.
- コントローラ領域 (Controller Area):** A green label above the code editor.
- エディタ領域 (Editor Area):** A green label below the code editor.
- 出力領域 (Output Area):** A green label at the bottom of the code editor, showing the execution output and any exceptions.
- リビジョン一覧領域 (Revision List Area):** The top-right section, highlighted in blue, shows a list of revisions (1 to 10) with their respective timestamps and status (save, compile, run).
- 差分表示領域 (Diff Display Area):** The bottom-right section, highlighted in orange, displays the source code diff and output for the selected revision (ID 4).

The code editor shows the following Java code:

```
1 class Assignment{
2     public static void main(String[] args){
3         String str = "12465244258209";
4         int x;
5         String str1;
6         String str2;
7
8         str1 = str.substring(0,3);
9         str2 = str.substring(6,9);
10
11         x = Integer.parseInt(str1 + str2);
12
13         System.out.println(x);
14     }
15 }
16
```

The output area shows the following exception:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "6524425820965244258209"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:583)
    at java.lang.Integer.parseInt(Integer.java:615)
    at Assignment.main(Main.java:11)
[Compiling and Running...]
124442
```

The diff display area shows the source code diff for revision 4 (save) and the output diff for revision 7 (run).

Source Code(Diff) for ID 4 (save):

```
class Assignment{
    public static void main(String[] ar
        String str = "12465244258209
        int x;
        String str1;
        String str2;
```

Output(Diff) for ID 7 (run):

```
str1 = string.substring(0,3);
str2 = string.substring(6,9);

x = Integer.parseInt(str1 + str

System.out.println(x);
```

Source Code(Diff) for ID 6 (compile):

```
String str1;
String str2;

System.out.println(x);
}
```

Output(Original) Compile error! for ID 6 (compile):

```
Main.java:10: error: variable x migh
System.out.println(x);
^
1 error
```

Source Code(Diff) for ID 10 (run):

```
class Assignment{
    public static void main(String[] ar
        String str = "12465244258209
        int x;
        String str1;
        String str2;
```

Output(Diff) for ID 10 (run):

```
124442
```

Change content overview area (変更内容一覧領域) is located at the bottom right.

探索的プログラミングが行われた箇所の特定

- ソースコードの編集履歴に着目し、探索的プログラミングが行われている**箇所**を特定する
 - どのプログラミングの機能に対してか(条件分岐、繰り返しなど)
 - どれほどの規模か(複数行、単一行、行内の要素)
 - どの深度において実施されているか(深度：ネストの深さ)
- 各学生の特定の課題に対する**躓きの原因**を教員側から推測することができる
 - 文法エラーかあるいはアルゴリズムの理解不足か

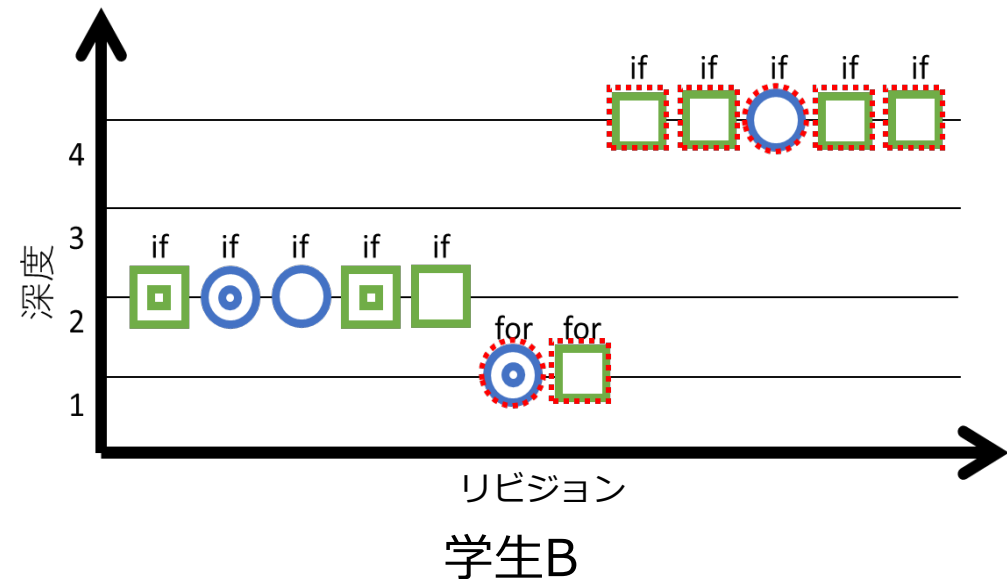
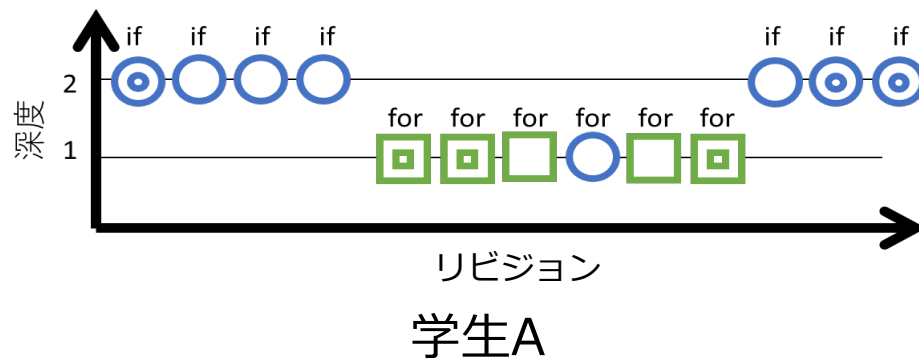
課題に行き詰まっている学生や間違えた方向に進む学生の発見・指導も容易になる

調査



- 提案手法で特定された探索が行われている箇所の情報が、実際の演習において教員が主体的に学生の躰きについて把握、アドバイスするために有用であるか調査を行った
- 大阪府立大学工業高等専門学校本科2年生41名が参加するC言語演習において得られたプログラミング行動のログを使用する
- うるう年を判定するプログラムを作成する課題において得られたプログラミング行動のログから、探索的プログラミングが行われている箇所を特定して教員に提示する
 - うるう年の判定には3つの条件がある
 - if文を入れ子にする、あるいは論理演算子で条件を組み合わせることが想定される

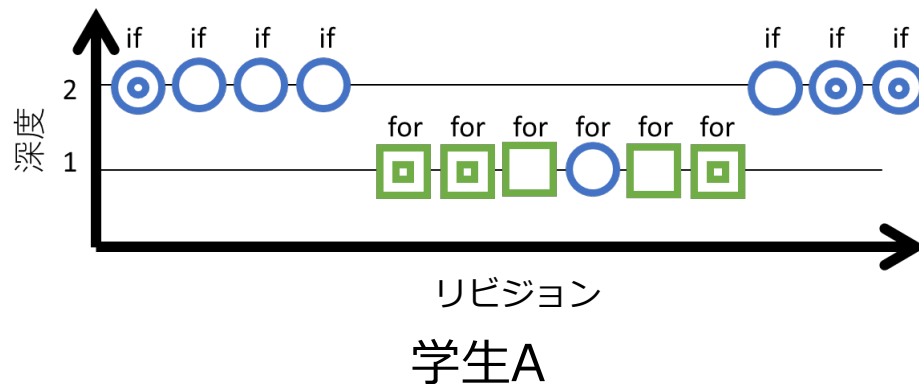
探索的プログラミングが行われた箇所の提示の例

○ 1行内の修正 □ 複数行の修正 ● 複数箇所の修正 ○ エラーが生じたリビジョン



探索的プログラミングが行われた箇所の提示の例



○ 1行内の修正 □ 複数行の修正  複数箇所の修正  エラーが生じたリビジョン



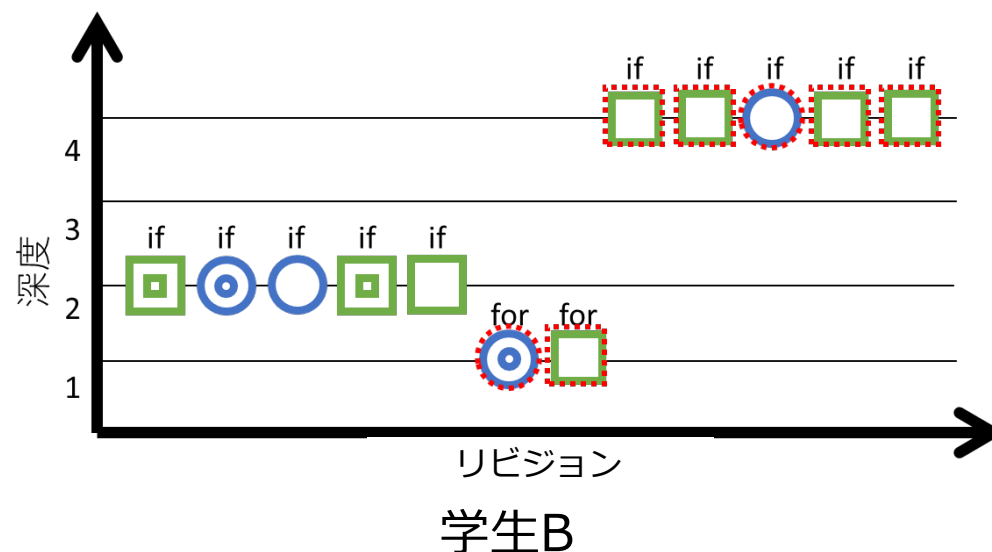
```
...  
if (year % 400 == 0 || year % 4 == 0  
    && year % 100 != 0) {  
    printf("うるう年です\n");  
}  
...
```

学生A：論理演算子によって条件を組み合わせたが、思い通りの出力が得られず条件式やif文の中の探索を続けていた

探索的プログラミングが行われた箇所の 提示の例

○ 1行内の修正 □ 複数行の修正  複数箇所の修正  エラーが生じたリビジョン

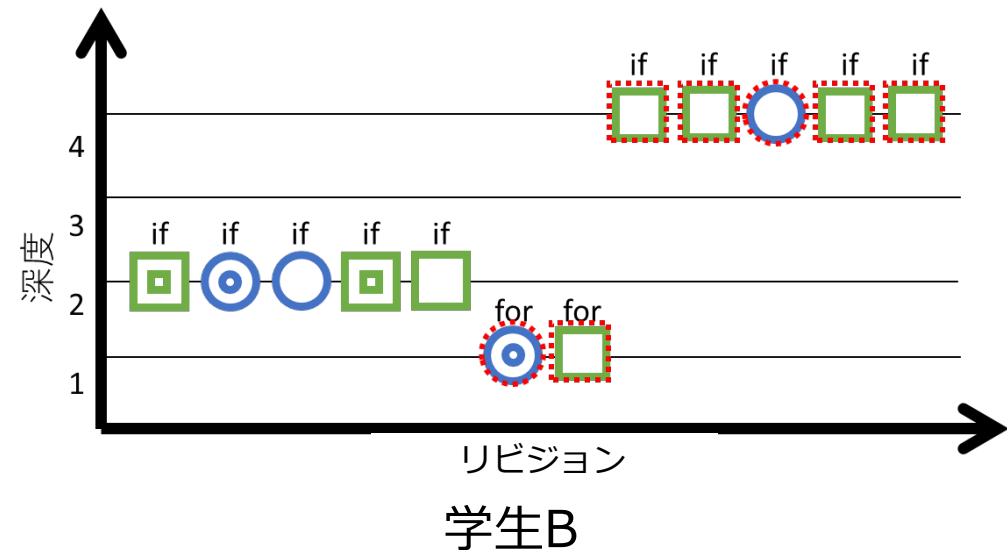
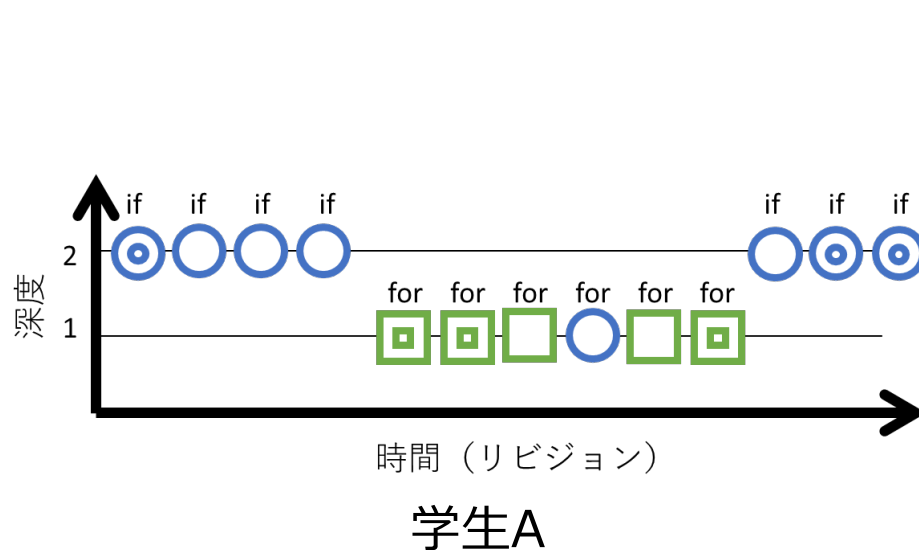
```
...
if (year % 4 == 0)
if (year % 100 == 0)
    if (year % 400 == 0) {
        printf("うるう年です\n");
    }else{
        printf("うるう年ではありません\n");
    }
}
```



学生B：論理演算子で繋げていた条件を**3つのif文に分解した際に、コンパイルエラーが生じ、エラーを放置して実装を続けていた**

探索的プログラミングが行われた箇所の提示の例

○ 1行内の修正 □ 複数行の修正 ● 複数箇所の修正 ○ エラーが生じたリビジョン



- 学生の試行錯誤に則ったアドバイス
- エラーの原因となった修正の確認・特定
- 各学生の理解度や苦手分野に応じた課題の配布

研究テーマの紹介

- 探索的プログラミングに着目した研究：
 - [T1]プログラミング演習における各学生の進捗状況を教員が把握するための支援環境の提案・開発
 - [T2]専攻分野、プログラミング言語、プログラミングの経験年数などが探索的プログラミングに及ぼす影響の調査
 - [T3]他デバイス（視線など）や色の情報を加味した探索的プログラミングやプログラミング理解に関する分析・調査
- プログラミング演習の支援に関する研究：
 - [T4]ソースコードの類似度に基づいたペアプログラミングの提案
- プログラミング環境の開発に関する研究：
 - [T5]Eclipse Cheを利用したプログラミング演習環境の提案・開発
 - [T6]対話型プログラミング環境の提案・開発(仮)
 - [T7]タブレットやスマートフォンでのプログラミング環境の開発(仮)