How should we teach debugging to secondary school students?

Laurie Gale

March 2024
Learning to program \rightarrow dealing with errors
Why research K-12 debugging?

Computing programmes of study: key stages 3 and 4

National curriculum in England

- use two or more programming languages, at least one of which is textual, to solve a variety of computational problems;

Different context to university → different challenges

It’s the curriculum → students don’t opt in
The challenges of learning to debug in K-12

Part I: Debugging behaviours
- Cognitive challenges
- Affective challenges
- Practical challenges

Part II: Attitudes and Emotions

Part III: Teacher’s Perspectives
Part I: Debugging Behaviour
Learning to debug is **HARD**

- Lots to store in working memory when learning to program\(^2\)
- Errors may be based on misconceptions
- Error messages are difficult for novices to read\(^3\)
- Novices tend to use ineffective debugging strategies\(^4,5,6\)
What we know about debugging behaviour

✓ Undergraduate studies in block-based and text-based programming languages
✓ K-12 studies in block-based programming languages

✗ K-12 studies in text-based programming languages
Study 1A

What *behaviours* for lower secondary students exhibit when *debugging*?
Study 1A

75 students (12–14-year-olds) from two schools

Debugging exercises

346 attempted exercises

≈7,000 snapshots

Qualitative content analysis
The debugging exercises

Programming Exercise 3

This program checks if someone should apply to be a computing teacher using the steps below:

- Input the user's age.
- Input the user's response to the question "Do you have a passion for teaching computing? Enter 'yes' or 'no': "
- If the user is 21 or over and does have a passion for teaching computing, the check should be a success. Otherwise, the check should be unsuccessful.
- Print the result of the check.

This program has 4 errors - have a go at fixing them all.

```python
# Question 3
print("This program will check if you should apply to be a computing teacher")
age = int(input("What is your age? "))
computing_degree = input("Do you have a passion for teaching computing? Enter 'yes' or 'no': ")

if age >= 21 and computing_degree == "yes":
    allowed_to_apply = "Successful"
else:
    allowed_to_apply = "Unsuccessful"

print("Result of check:",allowed_to_apply)
```

This program will check if you should apply to be a computing teacher
What is your age? 21
Do you have a passion for teaching computing? Enter 'yes' or 'no': yes
Result of check: Successful
Qualitative content analysis ... on log data?

Manually inspected log data → Examined changes made between runs → Generated codes/categories representing common behaviours
Some caveats

- Students were debugging “foreign code”
- They’d never used this code editor before
- The categorisation was binary and not ordered
Results

Six main themes:

1. First change
   - Ran code before making changes
2. Introduced additional errors
3. Resolved errors
4. Inconsequential changes
5. Positive debugging behaviours
6. Miscellaneous changes
Introducing some students

Alessia (struggled)
Perceived performance: 3/5
“I ran the code so I could see where and what line was wrong”

Gabriel (successful)
Perceived performance: 4/5
“Looked through line by line and used the error message”
Exercise 3

Programming Exercise 3

This program checks if someone should apply to be a computing teacher using the steps below:

- Input the user's age.
- Input the user's response to the question "Do you have a passion for teaching computing? Enter 'yes' or 'no': "
- If the user is 21 or over and does have a passion for teaching computing, the check should be a success. Otherwise, the check should be unsuccessful.
- Print the result of the check.

This program has 4 errors - have a go at fixing them all.

```python
1  # question 3
2  print("This program will check if you should apply to be a computing teacher")
3  age = int(input("What is your age? "))
4  computing_degree = input("Do you have a passion for teaching computing? Enter 'yes' or 'no': ")
5  if age > 21 or computing_degree == "yes":
6     allowed_to_apply = "Successful"
7  else:
8     allowed_to_apply = "Unsuccessful"
9  print("Result of check", allowed_to_apply)
```
Alessia

Perceived performance: 3/5
“I ran the code so I could see where and what line was wrong”

```
# Question 3
print("This program will check if you should apply to be a computing teacher")
age = int(input("What is your age? ""))
computing_degree = input("Do you have a passion for teaching computing? Enter 'yes' or 'no': ")

if age > 21 or computing_degree == "yes":
    allowed_to_apply = "Successful"
else:
    allowed_to_apply = "Unsuccessful"

print("Result of check:", allowed_to_apply)
```
Gabriel

Perceived performance: 4/5
“Looked through line by line and used the error message”

<table>
<thead>
<tr>
<th>Time of starting exercise: Mon, 27 Feb 2023 10:22:15 GMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time spent on exercise: 2 minutes 26.856 seconds</td>
</tr>
<tr>
<td>Run number: 1 out of 9 attempts on exercise</td>
</tr>
<tr>
<td>Time between last run: 27.925 seconds</td>
</tr>
<tr>
<td>Time spent on exercise 'so far': 27.925 seconds</td>
</tr>
<tr>
<td>Ran successfully? Yes</td>
</tr>
<tr>
<td>Error message:</td>
</tr>
</tbody>
</table>

```python
# Question 3
print("This program will check if you should apply to be a computing teacher")
age = int(input("What is your age? "))
computing_degree = input("Do you have a passion for teaching computing? Enter 'yes' or 'no': ")

if age > 21 or computing_degree == "yes":
    allowed_to_apply = "Successful"
else:
    allowed_to_apply = "Unsuccessful"
print("Result of check:",allowed_to_apply)
```
A comparison of their behaviours

- Ran code after 22 seconds
- (Potentially) used error message for guidance
- Made no corrective changes
- Added several syntax errors
- Ended with 3 syntax errors and 3 logical errors

- Ran code after 26 seconds
- (Potentially) resolved logical errors through testing
- Made several corrective changes
  - Added one syntax error, which they resolved straightaway
- Ended exercise in correct state
What’s stopping more students debugging successfully?

1. Knowledge of Python syntax
2. Time taken to get program successfully executing
3. Affective factors

Fragile knowledge

“Knowledge that is partial, hard to access, and often misused”7, p.4
How can this inform practice?

Some suggestions:

• More explainable error messages\textsuperscript{3,8}
• Tooling to help with syntax errors\textsuperscript{9}
• Teaching effective debugging strategies\textsuperscript{10,11}
• Discouraging ineffective ones
Part II: Attitudes and Emotions Towards Debugging
“Lacking a ready answer to the difficulty, the student not only feels at a complete loss, but is unwilling to explore the problem any further”\textsuperscript{4}, p.42
“The consequences that resulted after encountering the error seems to reflect the struck by lightning experience as well. ... These experiences left students puzzled, confused, frustrated, overwhelmed, and annoyed”12, p.81
Struggling when debugging

“The majority of students look horrified, put their hands up and say ‘that’s red, there’s a mistake’, and expect the teacher to present the solution to them” 13, p. 1034
“Every time after I type code and I run it for the first time, I expect it to fail. So that’s why ... it didn’t affect me that much either way” 14, p. 115
Learning to debug is *EMOTIONAL*

Feelings of frustration, anguish, and denial\[^{12,15}\]

Evidence of physiological reactions to error struggles\[^{14}\]

Negative emotional experiences contribute to negative attitudes\[^{16}\]

**Affective challenges**
An interaction with a student

“*I hate* computing.”

“Why’s that?”

“Because *I can’t do any of it.*”
Study 1B

To measure lower secondary students’ **thoughts** and **feelings** towards debugging
Study 1B

75 lower secondary students from two schools

Debugging exercises
346 attempted exercises ≈7,000 snapshots

Survey
Survey response data 73 responses

Correlational analysis
The survey

Perceived performance

“I feel I performed well on the exercises.”
(scale of 1-5)

Free text questions

“Why do you feel you performed this well?”

“What techniques did you use to find and fix the errors in the programming exercises?”

Construct statements

Attitudes:
• Self-efficacy
• Usefulness
• General perceptions

Emotions:
• Anxiety
• Frustration
• Joy

Demographic information

School
Year Group
Gender
Results – Correlation between attitudes

- Correlation between attitudes includes:
  - Self-efficacy
  - Anxiety
  - Frustration
  - Perceived performance
  - Usefulness

Gender is treated as a binary variable where:
- Male = 1
- Female = 2

Correlation coefficients:
- Gender to Self-efficacy: -0.42
- Self-efficacy to Perceived performance: 0.87
- Anxiety to Frustration: 0.68
- Perceived performance to Usefulness: 0.39
- Frustration to Usefulness: 0.29

These correlations indicate the strength and direction of the relationships between the variables.
“What techniques did you use to find and fix the errors in the programming exercises?

e.g. I searched the internet for a solution, I ran the code to see what errors it had, etc.”

Themes gathered using qualitative content analysis
Reported Debugging Behaviours

1) Running of code – 57 mentions
   Initial running of code (for purpose of reading error messages) – “I ran the code to see what and where the errors were”
   Repeated running of code – “I ran the code many times and made slight adjustments”

2) Inspection of code – 25 mentions
   General inspection of code – “[I] looked for obvious errors e.g. incorrect indent”
   Inspecting particular lines of code - “check the lines of code where a bug is more likely first”

3) Use of external resources – 12 mentions
   Use of cheat sheet
   Searching the internet

4) Trial and error – 10 mentions
   “[I] just kept going till the program worked”
What does this mean for lower secondary learners?

1. Attitudes and emotions must be considered when teaching debugging

2. Important for students’ debugging self-efficacy to be strong

3. Effective debugging behaviours should be taught to students
Part III: The Teacher Perspective
The challenges of learning to debug in K-12

Part I: Debugging behaviours

- Cognitive challenges
- Emotional challenges
- Practical challenges

Part II: Attitudes and Emotions

Part III: Teacher’s Perspectives
Learning to debug is **TOUGH FOR TEACHERS**

- Teachers aren’t always confident programmers\(^\text{17}\)
- Teachers rushing around the classroom\(^\text{13}\)
- Limited classroom time for teaching programming

**Practical challenges**
Study 2

What are lower secondary computing teachers' experiences and perspectives relating to teaching debugging?
Study 2

10 lower secondary computing teachers

Teacher interviews

35-65 minutes in length

Reflexive thematic analysis
The Interview Procedure

1. Triangulation with student data
   - Common errors and reactions
   - Common debugging strategies

2. Student-teacher interaction
   - Typical support provided
   - Change in debugging over time

3. How can research help?
Some contradictory findings:

On the barriers to syntax errors:
• “So we don't get that many problems in terms of syntax errors. They're relatively easy for them to, to find.”
• “we didn't notice any particular frustration or any emotional reaction at the beginning stages with syntax errors.”

On debugging strategies:
• “I think they fairly quickly learned that your error might not be in the line the errors directed at, so that is definitely one thing”
• “they are trained to use the debugging tools in Thonny, which are really, really good”
And some additional findings:

On emotional reactions:
• “you will hear often a loud exclamation of where you know something isn't working.”
• “The kind of raised fists in the air, it genuinely happens, and it's really exciting and we celebrate that”
How can this inform practice?

Main problems with K-12 debugging:
- Catering for diversity in ability
- Not helping every student
- Helping students stay motivated

How research could help:
- Concrete debugging teaching strategy
- Set of debugging exercises for common errors
Some Takeaways
Lessons Learnt

Study 1A
• Students often exhibit ineffective debugging behaviours
• E.g. repeated runs, tinkering
• Hard to resolve errors once these have begun
Lessons Learnt

Study 1B
• Attitudes and emotions to debugging are interlinked
• Self-efficacy is particularly interlinked
Lessons Learnt

Study 2

• Teachers can’t help every student to debug
• “A more scalable model is needed”
• Teachers with debugging scaffolding in place fare better
How can research help?

Informative error messages
Syntax-related tooling
Debugging-specific pedagogy

Cognitive aids
Affective aids
Teacher aids

Positive error culture
Initial success with bug-fixing

Debugging-specific pedagogy
Debugging-specific resources
Some questions to ponder

How do these findings compare to your experiences with novice debugging?

How do we best teach debugging within introductory programming? Separate lessons? Tooling?

What role does GenAI have to play in all this?
Thanks for listening!

Any questions?
References


A thanks to FLATICON for the icons used in the presentation