Programming Strategically

Thomas LaToza
https://cs.gmu.edu/~tlatoza/

@ThomasLaToza @ThomasLaToza@hci.social
The way the game is supposed to work is that the snake moves up, down, left, and right (using the keyboard). Every time the snake eats a dot, it grows in length by one. If the snake collides with itself, the game is over.

As you'll see when you play the game, the snake does not move up, down, left, and right. It just seems to move diagonally, and when you press the arrow keys in certain directions, the game ends.
Find an event immediately before the incorrect behavior
Trace control forwards, observing each statement until something incorrect happens
Find the statement that generated the incorrect output
Keep following the data used backwards until you find something that's wrong
guess and check
backwards search
forwards search
read the docs
check StackOverflow
ask a coworker
draw a whiteboard diagram
guess and check
backwards search
forwards search
read the docs
check StackOverflow
ask a coworker
draw a whiteboard diagram
programming strategy  a procedure for accomplishing a programming task

Goal: Fix issue

Where is the defect?

Subgoal

Subgoal

Subgoal

Trace output backwards

Which function generated the incorrect output?

Subgoal

Subgoal

Subgoal
programming strategy a procedure for accomplishing a programming task

Developers work more systematically and efficiently when given effective explicit programming strategies.

“Strategies determine success more than does the programmer’s available knowledge.”

“Experts seem to acquire a collection of strategies for performing programming tasks.”


better CSS debugging strategy
Q: Was function F’s implementation the ideal design, a hack, or accidental?

Strategy for answering:

1. Begin procedure `RetrieveRationaleFromCode`
   a. Initialize an empty set of rationales R
   b. For each comment in the function:
      i. If the comment provides information about the rationale for the implementation
         1. Add the rationale to R
   c. If R is non-empty
      i. Synthesize the rationales in R into an answer to the question.
      ii. If you successfully synthesized the rationales
           1. Stop, you have an answer.
   d. This strategy failed. Begin procedure `RetrieveRationaleFromDevelopers`

2. Begin procedure `RetrieveRationaleFromDevelopers`
   a. Initialize an empty set of developers D
   b. Use version control (e.g., git blame) to identify the developers in the entire history of the function who wrote or modified code, adding each developer to D
   c. Use your organization’s default communication channels (e.g., email, IRC, Slack), writing a message to everyone in D asking Q
   d. Wait until:
      i. Someone in D responds with the answer, then stop, or
      ii. All in D respond without the answer, or
      iii. You cannot wait any longer.
   e. This strategy failed. Begin procedure `InferRationaleFromCode`.

3. Begin procedure `InferRationaleFromCode`
   a. Fully comprehend the behavior of F at the level of computation
   b. Infer the intraprocedural intent of F, understanding how F interacts with all of the functions that call it and all of the functions that it calls.
   c. Using the intraprocedural intent of F, infer the possible architectural intents of F.
   d. Estimate the likelihood of each possible architectural intents of F. Which intent is most likely given the intents of the intraprocedural intent of F and the architectural intent of the software?
   e. Select the intent with the highest likelihood, and stop.
   f. If you were unable to infer intents, this strategy failed.
Please select your Strategy

```javascript
this.setState(state {
  strategyId: response.data,
  status: "saved"
});

localStorage.clear();

console.log("trash");

updateKnowledge = (array) => {
  let uniqueItems = new Set();
  array.forEach(item => {
    uniqueItems.add(item.toLowerCase())
  });

  this.setState(state {
    requiredKnowledge: Array.from(uniqueItems)
  });

  localStorage.setItem("new_requiredKnowledge", JSON.stringify(Array.from(uniqueItems)));
}

updateTools = async (array) => {
  let uniqueItems = new Set();
  let formattedAllTools = {};

  Object.values(this.state.allTools).forEach(item => {
    formattedAllTools[item.toLowerCase()] = items;
  });

  for (let i = 0; i < array.length; i++) {
    let item = array[i];
    let key = item.toLowerCase();
    if (formattedAllTools[key]) {
      uniqueItems.add(formattedAllTools[key]);
    } else {
      let success = true
      try {
        await axios.put("/dataManagement/technologies", {
          name: item
        })
      } catch (error) {
        console.error("There was an issue updating the tool:", error)
      }
    }
  }
```
STRATEGY :: strategy IDENTIFIER (IDENTIFIER+) STATEMENTS

STATEMENTS :: STATEMENT+

STATEMENT :: * (ACTION | CALL | CONDITIONAL | FOREACH | ASSIGNMENT | RETURN )+

ACTION :: (word | IDENTIFIER)+ .

CALL :: do identifier ( IDENTIFIER* )

CONDITIONAL :: if QUERY STATEMENTS

FOREACH :: for each IDENTIFIER in identifier STATEMENTS

UNTIL :: until QUERY STATEMENTS

ASSIGNMENT :: set IDENTIFIER to QUERY

RETURN :: return QUERY

QUERY :: (word | IDENTIFIER | CALL)+

IDENTIFIER :: ' identifier '
ASSIGNMENT :: set IDENTIFIER to QUERY

SET 'conflictedFiles' TO the project files that have a conflict

FOREACH :: for each IDENTIFIER in identifier STATEMENTS

FOR EACH 'file' IN conflictedFiles'
If you've spent a lot of time debugging unfamiliar code, the way that you probably debug is to first look at the failure, then look at the code to understand how it's architected, and then look for possible reasons for why the program failed. Once you have a guess, you probably then check it with things like breakpoints and logging. This strategy often works if you can have a lot of prior experience with debugging and inspecting program state. But if you don't have that experience, or you happen to guess wrong, this approach can lead to a lot of dead ends.

The strategy you're about to use is different. Instead of guessing and checking, this strategy involves systematically working backwards from the code that directly caused the failed output to all of the code that caused that failed output to occur. As you work backwards, you'll check each statement for defects. If you work backwards like this, following the chain of causality from failure to cause, you will almost certainly find the bug.

STRATEGY debug()

This first step will give you enough familiarity to find lines in the program that create the program's output. Read the names of all of the functions and variables in the program. Some programs produce command line output with print statements. Is the faulty output you're investigating printed to a command line?

If the faulty output is logged to a command line

To find print statements, try searching for keywords related to 'log' or 'print'

SET outputLines TO the line numbers of calls to console logging functions

Graphical output includes things like colored lines and rectangles

IF the faulty output is graphical output

To find these lines, try searching for keywords related to graphical output, like 'draw' or 'fill'. Focus on lines that directly render something, not on higher-level functions that indirectly call rendering functions.

SET outputLines TO the line numbers of function calls that directly render graphics to the screen

Now that you have some lines that could have directly produced the faulty output, you're going to check each line, see if it executed, and then find the cause of it executing. If you're lucky, you only have one output line to check.

FOR EACH 'line' IN 'outputLines'

First, let's make sure the line executed. You want to be sure that this is actually the
### Strategy: Design task

<table>
<thead>
<tr>
<th>Method</th>
<th>Self-guided</th>
<th>Guided</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template</td>
<td>4/14 (29%)</td>
<td>0/14 (0%)</td>
</tr>
<tr>
<td>Found and used example code as a template for implementation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decompose</td>
<td>9/14 (64%)</td>
<td>0/14 (0%)</td>
</tr>
<tr>
<td>Analyzed functional requirements for sub-problems, implementing each independently</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDD</td>
<td>2/14 (14%)</td>
<td>11/14 (79%)</td>
</tr>
<tr>
<td>Translated functional requirements into test cases, identifying sub-problems from test case requirements.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Strategy: Debugging task

<table>
<thead>
<tr>
<th>Method</th>
<th>Self-guided</th>
<th>Guided</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guess &amp; check</td>
<td>4/14 (29%)</td>
<td>0/14 (0%)</td>
</tr>
<tr>
<td>Participants found suspicious lines of code, modifying them and checking the effects of their modification.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forward search</td>
<td>9/14 (64%)</td>
<td>0/14 (0%)</td>
</tr>
<tr>
<td>Participants identified where the program began processing input, following its execution forward</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backward search</td>
<td>2/14 (14%)</td>
<td>11/14 (79%)</td>
</tr>
<tr>
<td>Participants identified faulty output and worked backwards through control and data flow dependencies</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Design task

1.30 times more likely to make more progress

\[ p < 0.023^* \]

Debugging task

1.96 times more likely to make more progress

\[ p < 0.004^* \]

are programming strategies tacit?
**STRATEGY FixCss(buggedElement)**

# You can use filter input to search for it
# Or you can scroll through the styles manually
Search through the stylings to find where it gets its undesired value

SET 'undesiredStyling' TO the line number and css file found in the search

IF 'undesiredStyling' is not found
# You will find all stylings applied to the element here
# Once you found the stylings you were looking for
# You can click small arrow to jump to the place it gets its value

Click on Computed tab and use filter to search

SET 'undesiredStyling' TO line number found here

SET 'perfectStyleList' TO an empty list of css properties

UNTIL buggedElement has desired styling
# you can add or change different css styles to the element
# it then applies instantly to element stylings
Use element. Style to apply css to buggedElement
add the style protperty to 'perfectStyleList'

DO ApplyCssToElement(buggedElement, 'perfectStyleList')

**STRATEGY ApplyCssToElement(element, style)**

# Css rules are cascading. The one with most priority applies
# This is how priority gets evaluated
# !important | style="" | id selector | class attribute, psudo class selector | type selector and psudo element
# For easy explanation: use this url: http://qnimate.com/dive-into-css-specificity/
# Also if there are two css files having the same selector, the file placed last in order is evaluated

IF style has to be applied to only this element
# e.g. choose last css file in order, use id selector and so on
Use strongest selector, apply style to element
RETURN nothing

IF style has to be applied on many elements
use class selector, apply style to element
RETURN nothing
• Strategy-related
  • Generality
  • Ambiguity
  • Imprecise steps
  • Required tool use

"I used chrome but still I was not able to find the NET section to find the CSS component. It took me a long time to find the component."

• Mismatch between the level of knowledge assumed by the strategy and possessed by the user

### code interacting with framework

<table>
<thead>
<tr>
<th>Activity</th>
<th>Likelihood (odds ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>search online forum</td>
<td>3.84</td>
</tr>
<tr>
<td>create diagrams</td>
<td>0.51</td>
</tr>
</tbody>
</table>

- 3.84x more likely
- 0.51x less likely

feeling stressed / nervous (LVHA)

<table>
<thead>
<tr>
<th>Activity</th>
<th>Likelihood (odds ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add print statements</td>
<td>2.42</td>
</tr>
<tr>
<td>read surrounding code</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>(2.42x more likely)</td>
</tr>
<tr>
<td></td>
<td>(0.17x less likely)</td>
</tr>
</tbody>
</table>

feeling sad / depressed (LVLA)

experiment with edits

likelihood
(odds ratio)

0.09
(0.09x less likely)

feeling excited / enthusiastic (HVHA)

ask for help from a colleague

likelihood (odds ratio)

2.13

(2.13x more likely)

Takeaways
be more effective with

**metacognition**  be aware of your problem solving process
be more effective with

self-regulation monitor progress and use of time

(Robillard et al. 2004; Falkner et al. 2014)
be more effective with better strategies
be more effective with sharing strategies
be aware of impact of how you feel

feeling stressed / nervous (LVHA)

feeling sad / depressed (LVLA)

feeling excited / enthusiastic (HVHA)
participate in a programming strategies mentoring session

email tlatoza@gmu.edu
Programming Strategically

- metacognition: be aware of your problem solving process
- self-regulation: monitor progress and use of time
- better strategies
- sharing strategies
- affect: be aware of impact of how you feel

Thomas LaToza
tlatoza@gmu.edu
https://cs.gmu.edu/~tlatoza/

@ThomasLaToza @ThomasLaToza@hci.social

https://howtoo.herokuapp.com/