ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE
Paper ID #37424

Work-in-Progress: Using a Scavenger Hunt to Tackle Challenges of CS1: Computational Thinking, Analyzing Code, and Debugging

Stephany Coffman-wolph (Assistant Professor)

Dr. Stephany Coffman-Wolph received her PhD from Western Michigan University and is currently an Assistant Professor in the ECCS Department (in Computer Science) at Ohio Northern University. Previously, she worked at The University of Texas at Austin and West Virginia University Institute of Technology (WVU Tech). While at WVU Tech she was a founding member and faculty advisor of AWESOME (Association of Women Engineers, Scientists, Or Mathematicians Empowerment) at WVU Tech. She is actively involved in community outreach with a goal of increasing the number of women in STEM and creating effective methods for introducing young children to CS concepts and topics. Dr. Coffman-Wolph's research interests include: Artificial Intelligence, Fuzzy Logic, Software Engineering, STEM Education, and Diversity and Inclusion within STEM. While growing up in Michigan, she was a dancer (ballet, tap, and jazz) until she graduated high school and a competitive figure skating through her senior year of college. Her hobbies include: reading mystery and fantasy books, knitting, shopping (especially to add to her shoe collection), yoga, and disc golf.

Work-in-Progress: Using a Scavenger Hunt to Tackle Challenges of CS1: Computational Thinking, Analyzing Code, and Debugging

What is the function of a rubber duck in programming? The rubber duck can act as a sounding board for programmers to work through tricky concepts or complicated logic [1]. It is known that speaking code logic out loud is hugely beneficial - especially when stuck. The "rubber duck debugging" concept was created by Andrew Errington [2]. Debugging, computational thinking, and code analysis are essential concepts for developing into a good programmer.

The research question addressed in this paper is how do we instill this concept into our students? Especially introductory programming students who are often resistant to trying new things or debugging independently. Most introductory programming instructors watch students write lines and lines of code without compiling the code or arrive at the instructor's office needing help because "it is almost working except this one compile error," which once fixed unearths many logical errors.

Enter the rubber duck prize! To add fun to the introduction of the debugging concept, the students are sent on a scavenger hunt around the building where the class is held. Locations included on the path: the computer lab, the department office, the help desk location, and the dean's office – thus familiarizing first-year students with important locations within the building.

Each scavenger hunt clue is a small C++ program – usually less than one page and is provided to the students via a hard copy. The students act as the computer to analyze code and to help further develop their computational thinking skills. By going "old school" with paper questions and away from the compiler with no ability to just run the program, the students must work on a deeper thought process with a focus on understanding the specifics of the material. The output statements within the code provide the location of the next clue, with the final clue leading the students to the professor's office. This assignment is stacked with other lecture materials and lab assignments to create a deeper knowledge of debugging techniques.

This paper will cover background on key concepts discussed (rubber duck debugging, computational thinking, and code analysis), the importance of learning debugging techniques, the specifics of the Computer Science 1 (CS1) scavenger hunt, hints and tips for adapting this for other languages and courses, and hints and tips for creating an online version.

Computer Science 1 (CS1) Course

ECCS 1611, the CS1 course at Ohio Northern University (ONU), is a four-credit hour beginning C++ programming course that focuses on learning to program using C++. The course meets for a traditional three 50-minute lecture sessions and a 90-minute computer lab session each of the 15 weeks. ECCS 1611 is the first course in a two-part sequence - the second, a Java course, focusing on early data structures, Object-Oriented Programming, Graphical User Interface (GUI) development, advanced programming concepts, and skills needed by professional programmers. The CS1 Course is taken by majors (and minors) in computer science, data analytics, robotics, applied mathematics, computer engineering, and electrical engineering. The course is taken by students with a wide variety of previous experience – with the majority having little or no

programming experience Therefore, the course is designed assuming no programming experience. Each section of the course typically contains 30-35 students and there are up to three sections run each fall semester. The course has one or two scavenger hunts per offering.

Scavenger Hunt

To prepare for the scavenger hunt, the instructor will need to prepare several small programs to serve as clues. The output of each program is the location of the next clue. Before class begins, the instructor "hides" the clues at the various locations. Additionally, the instructor needs to put up error signs at locations included as wrong answers or for answers generated by common mistakes by the students (see Figure 1). The approximate prep time for this activity is about one hour (writing the code and printing the clues takes 30-40 minutes and 20-30 minutes to set up the clues).

The scavenger hunt begins in the classroom (or an agreed-upon location). Students form teams of 2-4 students and are provided with a hard copy of a small C++ program, using program commenting to explain the rules (see figure 2). The instructor ensures all teams begin and then "disappears" to the end location (usually their office or a favorite place to hold office hours). If an instructor is concerned that students might "cheat" or get lost, it is helpful to recruit colleagues to provide check-in points/stations with a selection of interactive questions to earn the clues. Additionally, this allows students to interact positively with faculty outside of their instructor. These positive engagements and associations with faculty are beneficial for first-year students and potentially enable the students to meet future instructors. Finally, the variety of locations allows first-year students to become familiar with the Ohio Northern University College of Engineering office locations and staff [12].

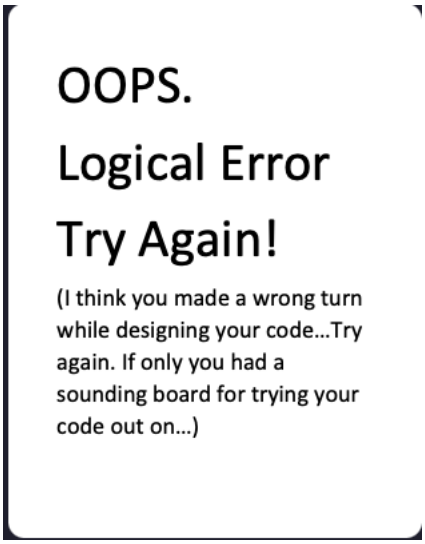

Figure 1: Logical Error Sign for Scavenger Hunt

```
int main()
{
    cout << "Hello Witches and Wizards! Welcome to the Halloween Scary Scavenger Hunt\n";
    cout << "Today we will be reviewing concepts we have been learning over " << endl;
    cout << "the past few weeks: basically a Midterm Exam Review! " << endl;
    cout << "(With a little fun thrown in :-) )" << endl;

    // Note: Every location is inside the Engineering (Kennedy) building
    // If your answer is not in this building, you made a mistake in your logic

    // Good Luck! May your brooms fly straight and your owls come home to roost.

    int tempVal = 1;
    int value1 = tempVal;
    int value2 = 0;
    int retVal = -99;
    value1++;
    value2 += value1;
    if (value2 % 2 == 0) { retVal = 0; }
    else { retVal = 1; }


    if (tempVal == retVal) { cout << "Dr. Stephany's Office\n"; }
    else { cout << "ECCS Department Office (2nd floor)\n"; }
    return 0;
}
```

Figure 2: Starting Clue with Instructions

Each scavenger hunt clue is a small C++ program – usually less than one page and is provided to the students via a hard copy (see figure 3 for an example clue). The students act as the computer to analyze code and to help further develop their computational thinking skills. Generally, the author uses the scavenger hunt to review material in preparation for the midterm exam. Therefore, the scavenger hunt covers a wide variety of topics, including (1) mathematical expressions, (2) mod operator, (3) integer math, (4) switch statements, (5) if statements, (6) increment/decrement, (7) for loops, (8) while loops, and (9) do-while loops.

```
/*
You are doing awesome so far!

You are getting soooo very close to the end

This next location should be familiar to you
 Especially for this class.

*/

#include <iostream>
using namespace std;

 int main() {
     int numb = 300;
     numb += 50;
     numb = (numb / 10) + 7;
     numb *= 100;
     numb = numb - 2 + 8 % 2;

     if(numb < 300) {
         if(numb < 200) {
             cout << "207 Computer Lounge\n";
         }
         else if(numb < 250) {
             cout << "205 Computer Teaching Lab\n";
```

Figure 3: Example Clue from Middle of Scavenger Hunt

The scavenger hunt ends at (generally) the professor's office or a place where office hours are held during the semester. The prize for completing the scavenger hunt is getting to select a rubber duck (see figure 4). Additionally, the students receive an explanation of the concept behind rubber duck debugging and how talking the code through with another person can be helpful. The students are directed to a website where they can read more details about the topic.

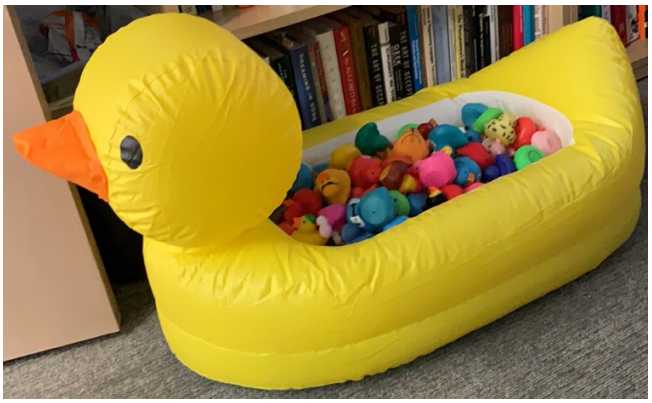

Figure 4: Selecting a Prize for Completing the Scavenger Hunt

Origin Story – Why Rubber Ducks?

When the author began offering the rubber duck as a prize, it was more of an accident than a planned effort. Rubber ducks were selected from hunting on Amazon for a reasonably priced bulk stress ball-like item that could be delivered in two days or less. Finding the rubber ducks was indeed a light-bulb moment for the author who remembered reading about "rubber duck debugging." The author selected the variety packs (e.g., sport-themed ducks, animal-themed ducks, non-yellow ducks, pirate ducks, camo ducks, etc.). The variety allows each student to select the duck that "quacks" to them (see figure 5 for the variety of ducks). (Surprisingly, many students spend several minutes contemplating and selecting their duck).

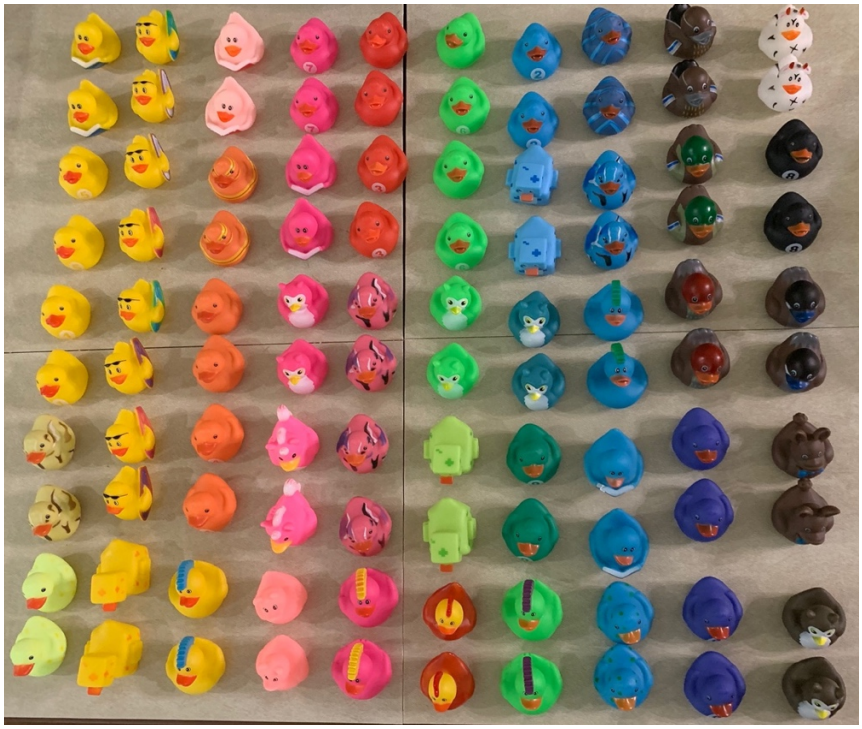

Figure 5: Example of the Variety of Rubber Ducks

Rubber Duck Debugging and Code Review

It is believed the reference of Rubber Duck debugging that brought this concept to most people's attention was "The Pragmatic Programmer: from journeyman to master" by Andrew Hunt and David Thomas [3]. Code review has been a long-standing software development practice for helping programmers improve the quality of their code [4]. The importance within the software development community has led to attempts to automate the process [4]. Studies have shown that code reviews from peers are helpful for learning concepts in introductory programming courses [5]. In rubber duck debugging, the rubber duck serves as a "peer." Rubber duck debugging assists on two measures. The first is slowing down and reading precisely what code has been typed (and not what you think you typed) [6]. The second is a shift in your thinking to that of the perspective of someone who does not know how to code or what is the problem the program is trying to solve [6]. Breaking down the problem and explaining it more thoroughly will often result in the programmer realizing their mistake, especially if it is a logic flow issue [6].

Alternative Versions – Online and Larger Class Sizes

An in-person scavenger hunt would be challenging for a hybrid or large class size. However, it would be easy to implement the scavenger hunt in an online version. (Additionally, the online version is adaptable for students with various accommodations). The clues would be posted on a Google Drive (OneDrive or similar) in PDF format. The answers would provide a link to the next clue (i.e., file names, clickable links, or QR codes [13]). The final clue could link to a letter, video by the professor, or an online Zoom (or Google Meet) session for wrap-up interaction. Students could be awarded a download of a duck picture or a file to 3D print a duck. As in the in-person version, the instructor could recruit colleagues to participate as check-in points via Zoom (or Google Meet) sessions.

Alternative Versions – Other Languages and Courses

This assignment is used in a C++ CS1 course with a mix of majors (computer science, computer engineers, electrical engineers, mechanical engineers, data analytics, applied mathematics, and others). However, this could easily be adapted by writing the clues in whatever language is being taught (C++, Java, C#, Python, R, etc.). Additionally, the clue programs could be made more challenging for an advanced class or honors course. The author has previously used this with a data structure and algorithm course and a 400-level programming languages course. For non-programming-related courses, this is easily adaptable for any type of problem with a numeric solution. The clue would be the problem, and the solutions would be the room/location numbers. The scavenger hunt can also be expanded to other types of solutions – adapting it into a multiple-choice problem with each answer having a location associated with it. For example, answer choice 'A - Polymorphism' would direct them to the library and answer choice 'B - Interface' to the student study lounge.

Difficulties, Strengths, and Weaknesses

Approximately half of the students make very few mistakes during the scavenger hunt. Most students find all six clues in about 30 minutes. The "wrong logic" signs at the incorrect locations

allow for graceful failure that the instructor does not have to know about – so it is difficult to judge precisely. A few students struggle, and many realize that reading code without the compiler to run the code is challenging. Additionally, the students are able to identify course concepts they do not entirely understand and can ask their peers to help explain.

The students enjoy getting out of the traditional classroom instruction. They learn to read code written by others and self-identify areas of confusion. The duck prizes at the end provide new information even to the experienced programmers in the course. Students have been known to bring their ducks to lab and exams as the semester progresses. (Additionally, students hold on to the ducks and carry them in their computer bags).

It is essential to collect the names of those who complete the activity and make the assignment worth points. Otherwise, students will give up partway through the scavenger hunt or just leave without participating. In addition, the in-person scavenger hunt works partly on the honor system - the student could just wander about the building until they find the designated number of sheets. (To avoid this, place a TA or faculty at each location or the majority of locations and ask to see the previous clue).

Computational Thinking

Computational thinking is defined as a technique to solve a problem effectively with a machine or computational tools (i.e., algorithmic thinking, decomposition, abstraction, pattern recognition) [7]. Computational thinking is an essential skill for programmers and a concept that can be difficult but essential to instill during an introductory programming course [8]. Furthermore, some would argue that these skills are critical for all engineering students with learning programming languages as an excellent vehicle to develop these skills and general problem-solving skills [8-9]. Students are told that all scavenger hunt locations are within the same building (narrowing the scope) and hints towards the correct answers. The scavenger hunt provides a fun and safe environment to practice these computation thinking skills – particularly decomposition and algorithmic thinking.

Code Analysis

Students need to improve their skills at reading, analyzing, and testing their code [10-11]. The first step of this progress is reading code and determining what the code does accurately. Students tend to rely heavily on the compiler to find the syntactical flaws and rely on running the code to determine the functionality of their code. Without an understanding of how a computer steps through code, it can be difficult to find logic errors. Moving to a hard copy of code forces students to do the analysis themselves – enter the scavenger hunt. The students know that the correct answer is obtainable, and they can learn through low stakes where failure can be quickly corrected (i.e., try another answer choice).

Future Work

When this project began, the author aimed purely for fun with a nice industry twist. However, when students started bringing their rubber ducks to exams, positive reviews of the scavenger

hunt from students, and previous students verified that the new first-years would be getting their ducks, the author realized this activity had greater impact. So, next year, the author will be applying for Institutional Review Board (IRB) Research Study and will survey current and former students regarding the impact of the rubber ducks on several points, including (1) use of the rubber duck in other classes, (2) do they still have the rubber duck, (3) increased sense of identity as a programmer, and (4) feeling of being part of the "rubber duck club" within the department. The instructor will also look at grades of programming assignments before and after the introduction of debugging in the course.

References:

[1] R. D. Debugging, "Rubber duck debugging," – *Rubber Duck Debugging – Debugging software with a rubber ducky*. [Online]. Available: https://rubberduckdebugging.com/. [Accessed: 08-Nov-2021].

[2] *Re: Not an AWK question*. [Online]. Available: http://lists.ethernal.org/oldarchives/cantlug-0211/msg00174.html. [Accessed: 08-Nov-2021].

[3] Andrew Hunt and David Thomas. 2000. *The pragmatic programmer: from journeyman to master*. Addison-Wesley Longman Publishing Co., Inc., USA.

[4] Ramachandran, L., & Gehringer, E. F. (2012, June), *An Automated Approach to Assessing the Quality of Code Reviews*Paper presented at 2012 ASEE Annual Conference & Exposition, San Antonio, Texas. 10.18260/1-2—20914.

[5] Brown, T., & Walia, G. S., & Radermacher, A. D., & Singh, M., & Narasareddygari, M. R. (2020, June), *Effectiveness of Using Guided Peer Code Review to Support Learning of Programming Concepts in a CS2 Course: A Pilot Study* Paper presented at 2020 ASEE Virtual Annual Conference Content Access, Virtual Online. 10.18260/1-2—34504.

[6] B Hayes, D., 2019. *Rubber Duck Debugging: The Psychology of How it Works*. [online] Thoughtful Code. Available at: <https://www.thoughtfulcode.com/rubber-duck-debugging-psychology/> [Accessed 12 February 2022].

[7] Cruz Castro, L. M., & Shoaib, H., & Douglas, K. A. (2021, July), *Computational Thinking Frameworks used in Computational Thinking Assessment in Higher Education. A Systematized Literature Review*. Paper presented at 2021 ASEE Virtual Annual Conference Content Access, Virtual Conference. 10.18260/1-2—36824.

[8] Brophy, S. P., & Lowe, T. A. (2017, June), *A Learning Trajectory for Developing Computational Thinking and Programming* Paper presented at 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. 10.18260/1-2—27472.

[9] Lee, S. B., & Lovvorn, H. (2016, June), *Building Computational Thinking Skills Using Robots With First-Year Engineering Students* Paper presented at 2016 ASEE Annual Conference & Exposition, New Orleans, Louisiana. 10.18260/p.26409.

[10] Stephen H. Edwards. 2003. *Improving student performance by evaluating how well students test their own programs*. J. Educ. Resour. Comput. 3, 3 (September 2003), 1–es. DOI:https://doi.org/10.1145/1029994.1029995.

[11] Qiu, T., & Feng, M., & Lu, S., & Li, Z., & Wu, Y., & Zoltowski, C. B., & Lu, Y. (2017, June), *Online Programming System for Code Analysis and Activity Tracking* Paper presented at 2017 ASEE Annual Conference & Exposition, Columbus, Ohio. 10.18260/1-2—28722.

[12] Gunn, C. (2008, June), *The Value Of Scavenger Hunts In The Life Of A Freshman* Paper presented at 2008 Annual Conference & Exposition, Pittsburgh, Pennsylvania. 10.18260/1-2--4477

[13] Coffman-Wolph, S., & Gray, K. (2020, June), *Computer Coding Scavenger Hunt Using Quick Response Codes (Resource Exchange)* Paper presented at 2020 ASEE Virtual Annual Conference Content Access, Virtual Online. 10.18260/1-2—34320.