Beacons and Novice Programming Comprehension

Mark Noone and Aidan Mooney [S, C]
mark.noone@mu.ie, aidan.mooney@mu.ie
Department of Computer Science
Maynooth University
Maynooth, Co. Kildare, Ireland

S = Senior Author, C = Corresponding Author

Abstract

Computer Science courses at tertiary level have one of the highest drop-out rates internationally. One of the main issues for this high attrition rate is often seen as CS1, the first Computer Science module usually encountered by students, which has a strong emphasis on computer programming. In order to aid students in the steep learning curve associated with programming, many different techniques have been utilised, to a varied degree of success. This paper aims to discover if particular lines of programming code exist that can help readers easily identify its functionality - referred to as a "beacon". In a program containing a sort function, for example, advanced programmers might observe the swap code inside a loop and comprehend that it is a sorting algorithm, and therefore a beacon, without much further examination.

This paper details the first phase of a study examining the presence of beacons in CS1 standard Java code using eye-tracking technology. In particular this paper will focus on the collection of data from non-novice programmers to determine whether or not beacons can be detected. Participants in this study were presented with basic Java programs and were asked to determine, from a list of possible options, what output was correct. Data was collected using an eye-tracking device during a phase of experimentation and this data was subsequently analysed. From the analysis we were able to detect that some beacons did exist in the code. In the future, some method of displaying these beacons could potentially be implemented as a form of intervention to aid students within the initial stages of learning a programming language.

Keywords

Eye-Tracking, Programming, Beacons, Comprehension, Computer Science Education

1. Introduction

This paper focuses on determining if "beacons" can be detected by participants while they attempt to comprehend Java code. According to Susan Wiedenbeck "beacons are lines of code which serve as typical indicators of a particular structure or operation" (Wiedenbeck, 1986). In other words, beacons are particular lines of code that can quickly help us identify the functionality of that code block. For example, in a sort program, advanced programmers might notice the swap code inside a loop and realise that the code is implementing a sorting algorithm. Wiedenbeck showed this in her aforementioned paper.

The primary goal of this study is to expand upon the work of Wiedenbeck as well as that of Hegarty-Kelly et al. (Hegarty-Kelly, Bergin, & Mooney, 2015) to determine if beacons exist in programming languages and code blocks, and, if so, to demonstrate that they are an important factor in understanding the functionality of that program. The overall aim is to prove the hypothesis for smaller blocks of code than originally used, such as a loop or an array manipulation.

In this study, eye-tracking data was collected from experienced undergraduate programmers while they solved programming problems. The goal was to detect potential beacons and determine if there is any further evidence of their existence. All data collection involved the use of a commercial eye-tracking device called the EyeTribe eye-tracker (since discontinued), which was a budget friendly 60Hz eye-tracking device.

Busjahn et al. have shown that studying a person's gaze can record fine grain information in space and time allowing for a deeper understanding of comprehension levels (Busjahn et al., 2014). With the examination and analysis of the collected eye-tracking data, we will attempt to determine if some areas of code are indeed beacons.

2. Motivation

The main motivation for this study relates closely to the analysis of Programming Comprehension. In Ireland, the failure rate for first year Computer Science students was recently reported at being around 25%, the highest among all disciplines in higher education (Harmon & Erskine, 2017). This is a serious issue that needs to be addressed. If the presence of beacons in programming code is verified they could be used to help focus learner's attention on particular code blocks as a form of intervention to reduce comprehension issues and in turn help lower dropout rates.

The field of eye-tracking has become more prevalent in recent years and there is growing interest in using eye-tracking within Computer Science Education. Eye-tracking is an important tool as it can tell us something about how people process information that they might otherwise be unable to vocalise. How do you make decisions when you are programming? It might be hard for you to explain, but your eyes and your cognition tell a part of the story. This is based on the eye-mind assumption asserted by Just and Carpenter (1980) , that the longer a fixation remains in an area the more cognitive processing is being undertaken.

3. Background

In addition to Wiedenbeck's (Wiedenbeck, 1986) work in Programming Comprehension, the early theory for beacons was formulated based on an experiment called "Duncker's radiation problem" (Grant & Spivey, 2003). The concept of this experiment was as follows.

A diagram, presented in Fig. 3.1 is shown to participants (medical students) which shows an inoperable tumour surrounded by healthy tissue. The participants are asked how they could cure this ailment. If a laser was directly fired at the tumour, healthy tissue would be damaged. According to the paper; "The solution requires firing multiple low-intensity lasers from several angles outside the healthy tissue so that they converge at the tumour, with combined intensity sufficient to destroy it."

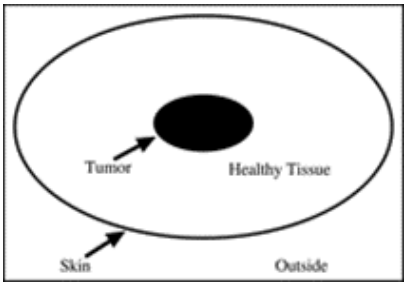

Figure 3.1 – Diagram of a tumour underneath healthy skin

The first time the experiment was ran (experiment 1), it was a base test where 14 medical students with no knowledge of the problem were asked to cure the ailment from the diagram with no assistance. While only 34% of participants succeeded in their task, by analysing the successful candidates' eye-tracking data, the experimenter learned that part of the skin was what they considered the "beacon" in the diagram.

In the second experiment, three different groups were tested. Group 1 repeated the experiment in the same way as phase 1. Group 2 were shown the diagram with the beacon animated (only animated slightly to be noticed at a cognitive level, imagine something pulsating very slowly with a small pulse radius). Group 3 were shown the diagram with the wrong area of the diagram animated.

The results of experiment 2 were as follows:

- Group 1 and 3 had approximately the same results as in the control group (experiment 1). That is, without animation of the beacon, and interestingly, when the wrong area of the diagram was highlighted, participants were only ~33% likely to give the correct answer.
- Group 2 however, were able to identify the solution statistically significantly more often. That is, when the beacon identified from the control experiment was highlighted, 67% of people successfully found the solution.

Based on the findings of this experiment it was hypothesised that beacons may exist when people comprehend code and that perhaps people who successfully comprehend code could aid those that do not understand it as readily. This is what Wiedenbeck's work related to and it is the aim of this paper to verify and extend upon her work.

Another key researcher in the area is Brooks whose top-down theories are hypothesis-driven. Brooks postulates that the programmer begins by making general hypotheses about the program's function and then looks for concrete evidence by scanning for beacons (Brooks, 1983). If found they may conclude that the hypothesis is correct, while if not found, they must revise/reject the hypothesis.

Soloway et al. (E Soloway & Ehrlich, 1984; Elliot Soloway, Adelson, & Ehrlich, 1988) built on this work providing a plan-based theory. Here, understanding begins with the programmer hypothesising a

high-level program goal, then decomposing it into sub-goals. The programmer utilises previously acquired schematic knowledge (plans) to satisfy the sub goals and ultimately the top-level goal.

Aschwanden and Crosby (Aschwanden & Crosby, 2006) and Crosby et al. (Crosby, Scholtz, & Wiedenbeck, 2002) also expanded on the work of Wiedenbeck. They evaluated the use of beacons in programming comprehension. In summary, experienced programmers appear to use beacons, novice programmers do not or if they do they are different to those of experience programmers. Poor naming conventions render novice programmers unable to comprehend code that they could comprehend when informative names are used.

It is clear that by analysing gaze during code reading processes, we will be able to learn a lot about how the participant thinks. A person's gaze gives a record of their visual behaviour on a super fine-grained scale, in both space and time. The observation of eye movements adds an objective source of information about programmer behaviour to the collection of research methods in computing education which can inform the teaching and learning of programming.

Focused attention is the ability to respond discreetly to specific visual, auditory or tactile stimuli. The overall goal of this study is to focus the attention of novice programmers while they perform programming comprehension tasks to facilitate improved comprehension. By visually highlighting critical areas of code blocks, a beacon, we hypothesise that a novice programmer will be able to solve programming comprehension tasks that they would not be able to solve ordinarily.

4. Experimental Phase

The study involved the planning, implementation of an experiment and the analysis of the collected data. Data was collected using the EyeTribe eye-tracker as well as through survey questions. The survey gathered participant demographic details such as their age, gender and level of programming experience. The experiment builder used to combine all components of the experiment was OpenSesame (Mathôt, Schreij, & Theeuwes, 2012). OpenSesame allows for direct links with eye-tracking devices allowing for everything to run smoothly.

A set of ten Java examples was developed for the experiment. Research of Wiedenbeck's paper (Wiedenbeck, 1986) and the Eye Movements in Programming (EMIP) Workshop data sets (Bednarik, 2014) provided a basis for the creation of two test questions. The remaining questions were developed entirely from scratch. They were based on simple programming concepts that are part of our CS1 curriculum. An example test case (which was modified from an EMIP example) can be seen in Fig. 4.1.

```
String text = "Hello World";
//Store an index
int positionW = text.indexOf("W");
int wLen = text.length();
String w = new String();
w = text.substring(positionW, wLen-1);
//Replace w substring in text with Sun
System.out.print(text.replace(w, "Sun"));
```

Figure 4.1 – Sample test case structure

For each question, a graphical representation of the test case was designed, developed and incorporated into the OpenSesame experiment. The concepts that were chosen to be elements in the test cases can be seen in Fig 4.2.

- Selection Statements (Q1, Q5)
- Loops (Q6, Q7)
- Switch Statements (Q2)
- String manipulation (Q3, Q4)
- Arrays (Q6, Q7)
- Nested Loops (Q8, Q9, Q10)
- Comments (Q3, Q4, Q6, Q7, Q8, Q9, Q10)

Figure 4.2 – List of Concepts within Experiment Questions

These concepts were chosen to ensure a variance in difficulty, time required to solve, mathematical and non-mathematical solutions. The aim was to get as much eye-tracking data from the questions as possible to see where people focus their attention and gaze when comprehending the code. We were also interested in the "easier" questions to see if the participants hone in on one particular spot immediately upon seeing the code block or if they still read through it linearly.

Once the experimental setup was complete, a pilot testing phase began with 5 subjects to verify that the data collection process was working correctly. The main experimentation phase then took place with twenty four participants completing the experiment, with each experiment taking approximately 30 minutes. Each participant's eyes were calibrated to the EyeTribe, they then completed the experiment and provided consent and survey answers both before and after the trials.

5. Results

Interpreting the eye-tracking data file is not easily done manually. A toolkit was to be used that would read in the raw data file, partition it based on trials (one per question, 10 questions per participant) and produce visual outputs that would make the results clearer to analyse. This tool is called PyGazeAnalyser

(Dalmaijer, Mathôt, & Van der Stigchel, 2013) and was written by a member of the OpenSesame team, Edwin Dalmaijer. It was built to directly interact with output files from eye-tracking devices.

Some modification was needed to get the example analysis script to work with this data set. Fig. 5.1 contains example representation of the four types of output files produced by this analysis script. These are:

- A fixation map, showing where the participants gaze rested for a period longer than half a second; a fixation. The bigger the circle, the longer the fixation.
- A heat map, showing the areas most fixated on. Heat maps are used to see how a participant's gaze is distributed over a stimulus. A heat map consists of different colours where red usually represents the highest number of fixations or the longest time, while green represents the least.
- A raw data file, simply showing all areas the participant rested their gaze on.
- A scan path, showing the participants saccades, the quick movements between fixations. By analysing a scan path we can understand which visual elements were fixated on the most by a participant and the order in which these occurred.

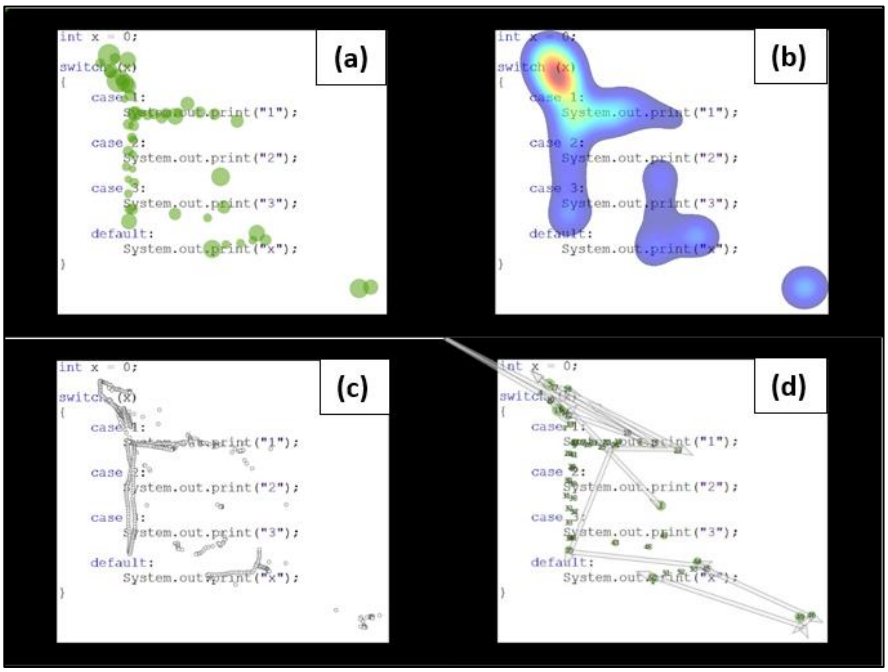

Figure 5.1 – (a): Fixations, (b): Heat Map, (c): Raw Data, (d): Scan Path

Some potential beacon information was discovered from this data. We determined that an area of code could be a beacon by examining the heat maps and the dwell time across participants. If the majority of participants who correctly answered a question spent a large percentage of their dwell time on an area (or went straight to an area of the code and proceeded to solve the question quickly) then it was marked as a potential beacon.

Some of the possible beacons determined were:

- Multiple if conditions within a question.
- Loop calculation points (e.g. ``result += array[i]'').
- Nested Loop bounds (participants tended to look at both loop bounds and the loop calculation frequently on nested loop questions).

Some visualisations of the kind of heat map outputs we used to determine the above beacons are shown in Fig. 5.2. Many participants had similar results for these questions. The findings from Phase 1 of the study helped to inform the questions and methodology that will subsequently be used in Phase 2. This was extremely beneficial as there was now knowledge of which beacons were potentially going to appear the most. We can now ensure that the Phase 2 questions have as many of these potential beacons present as possible for further verification.

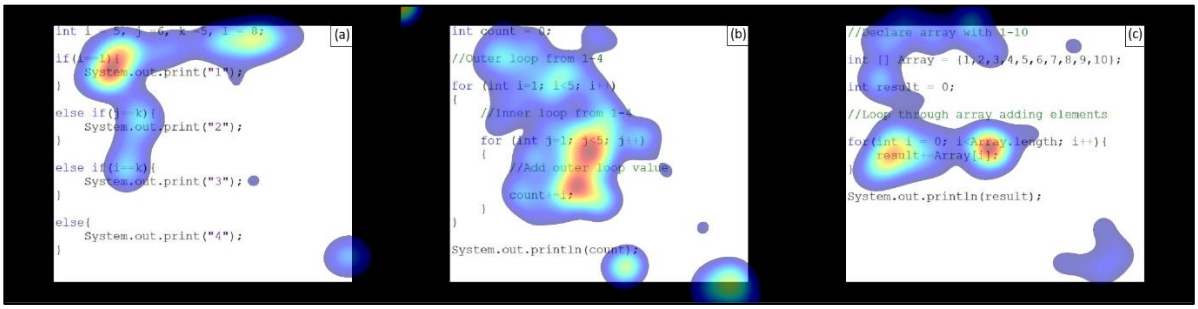

Figure 5.2 – Heat maps for (a): Conditions beacon, (b): Loop Calculation Point Beacon, (c): Nested Loop Bounds Beacon

6. Conclusions and Future Work

From the data gathered, some conclusions about the existence and efficacy of beacons can be reached. It has been shown that some elements of code do provide more clarity on the functionality of the overall program or method than other elements. We also discovered that students who get answers correct tend to spend most of their time focusing on these particular elements of code. This tells us that in our pedagogical approach, we might need to focus on these key elements of programming even more than we do.

In phase 2 of the study further verification will be done on the beacons found in phase 1. In particular, testing will be undertaken (again using experienced undergraduate programmers) on how we might display such a beacon to a user (without it being obvious), during a code comprehension task. The goal with displaying this beacon is for it to be picked up at a cognitive level rather than "seen". It should not be displayed for a longitudinal period of time. Additionally, larger group sizes need to be queried on their

opinions of beacons. We need to be certain these beacons do exist before we implement the display of them as a teaching aid.

The methodology for phase 2 will follow the structure of the Duncker's Tumour Experiment (Grant & Spivey, 2003). Namely, having a control group who are not presented with beacons, an experimental group who are presented with flashing beacons and a further experimental group who are presented with incorrect beacons. This will allow for complete cross-analysis to determine if the beacon is having any effect. Phase 2 will also make use of more advanced technologies than the EyeTribe, namely, the EyeLink 1000 Plus which is a 1000Hz industry standard eye-tracking device. This will provide much higher accuracy and precision for the collected data. Data analysis will be done using the EyeLink data viewer. This tool is both provided and recommended by the EyeLink developers for use with EyeLink output files.

After phase 2, work can begin on the overall goal of the project, which is to take the known beacons and use them to provide visual feedback (or subtle hints like in the Duncker's Tumour experiment) to struggling novice programmers. The hope is that with this visual feedback, students may discover their own mistakes which leads to more concrete learning and retention.

7. Acknowledgements

The authors would like to thank all experiment participants in both phases for their help and time. This work was assisted by funding from the John and Pat Hume scholarship, Maynooth University.

References

Aschwanden, C., & Crosby, M. (2006). Code Scanning Patterns in Program Comprehension. *Proceedings of the 39th Hawaii International Conference on System Sciences*. Retrieved from http://pdf.aminer.org/000/641/141/is_there_any_difference_in_novice_comprehension_of_a_small.pdf

Bednarik, R. (2014). Eye Movements in Programming Dataset 2014. Retrieved from http://emipws.org/datasets-2014/

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, *18*(6), 543–554. https://doi.org/10.1016/S0020-7373(83)80031-5

Busjahn, T., Schulte, C., Sharif, B., Simon, Begel, A., Hansen, M., … Antropova, M. (2014). Eye tracking in computing education. *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 3–10. https://doi.org/10.1145/2632320.2632344

Crosby, M. E., Scholtz, J., & Wiedenbeck, S. (2002). The Roles Beacons Play in Comprehension for Novice and Expert Programmers. In *14th Workshop of the Psychology of Programming Interest Group* (pp. 58–73).

Dalmaijer, E. S., Mathôt, S., & Van der Stigchel, S. (2013). PyGaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior Research Methods*, 1–16. https://doi.org/10.3758/s13428-013-0422-2

Grant, E. R., & Spivey, M. J. (2003). Eye movements and problem solving: Guiding attention guides thought. *Psychological Science*, *14*(5), 462–466.

Harmon, D. & Erskine, S.(2017). Eurostudent Survey VI. Dec. 2017.http://hea.ie/assets/uploads/2018/01/HEA-Eurostudent-Survey.pdf.

Hegarty-Kelly, E., Bergin, S., & Mooney, A. (2015). Using focused attention to improve programming comprehension for novice programmers. In *Third International Workshop on Eye Movemenets in Programming*.

Just, M. A., & Carpenter, P. A. (1980). A theory of reading: From eye fixations to comprehension. *Psychological Review*, *87*(4), 329.

Mathôt, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source graphical experiment builder for the social sciences. *Behavior Research Methods*, *44*(2), 314–324. https://doi.org/10.3758/s13428-011-0168-7

Soloway, E, & Ehrlich, K. (1984). Empirical studies of programming knowledge (Research Report# 16). *New Haven, CT: Yale University, Department of Computer Science Cognition and Programming Project*.

Soloway, Elliot, Adelson, B., & Ehrlich, K. (1988). Knowledge and Processes in the Comprehension of Computer Programs. In *The Nature of Expertise* (pp. 129–152).

Wiedenbeck, S. (1986). Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, *25*(6), 697–709. https://doi.org/http://dx.doi.org/10.1016/S0020-7373(86)80083-9