Marissa Walther
HCI
3/9/2020

Evaluation of Tools Designed for Novice Programmers to Help Them Learn

Abstract

Novice programmers, defined as students who are learning programming, require different tools than those who are experienced programmers. Experienced programmers have a familiarity with the basic concepts that are encountered in programming, and are able to use that knowledge to solve problems. Since an experienced programmer is able to understand these concepts, they are able to debug their programs faster and they understand what different error messages mean and how to fix them. To a novice programmer, a lot of these error messages are cryptic and hard to understand. To aid these novice programmers, there exist different tools that provide different resources aimed at helping to enforce an understanding with different programming concepts. This paper will evaluate how novices can interact with two such tools and then propose an alternative.

1. Introduction

Many of the tools that are used today for programming are designed for those who are considered to be experienced programmers. Experienced programmers, defined as those who have a base familiarity of the fundamental programming concepts, are able to use these tools in order to fix and adapt their code for different situations. These tools, range from the feedback messages that compilers return in response to errors to debuggers in different Integrated Development Environments (IDEs). They are mostly designed for experienced programmers. Novice programmers, when presented with the ability to use these tools, often have trouble interpreting messages and understanding how the information from these tools can be used to help further their knowledge of the coding problem. In an attempt to help these novices learn the fundamentals and understand how to fix the problems in their code, different types of tools were created to assist the novice rather than the experienced programmer. Some examples of these different tools include Omnicode and BlueFix.

Omnicode, an IDE developed specifically for novice programmers using Python, aims to give the user real time feedback about what different variables contain and functions output before the user hits run. The main goal for Omnicode is to present a visual representation of what the code is doing and how well it is performing at any point in the program.

BlueFix, an extension for BlueJ and Java, aims to give the user different layers of feedback based on how stuck the programmer seems to be. This feedback is altered by popular vote, where if the user finds that a certain message helped diagnose a problem then they would upvote or comment on it, allowing the messages that got the best scores to be the first things that the feedback system would provide.

This paper will evaluate how the students and novice programmers can interact with these three tools, and then suggest an alternative that combines the strengths of the different systems.

2. Omnicode

Omnicode, a live IDE developed by researchers at the University of California, contains a set of tools that are aimed at helping the novice programmer. One of the main tools that Omnicode has include scatterplots that visualize how the value stored in a variable changes over time or by different actions. Another tool is an overlay that when a user has selected a test case in a different menu, the overlay will show what the values are of a variable or the return value of a function by parsing the values at the different execution steps in the program. Omnicode's main goal is to use these tools to help a user have a better sense of visualizing what the program will do at any specific point.

2.1 Scatterplots

The scatterplots for Omnicode are presented in a side panel on the right side of the IDE. The side panel contains a visual list of graphs based on what is deemed relevant by Omnicode, but also allows for custom graphs to be requested by the user. The scatterplots show by default how a variable is changing based on the y axis by a certain event or time on the x axis.

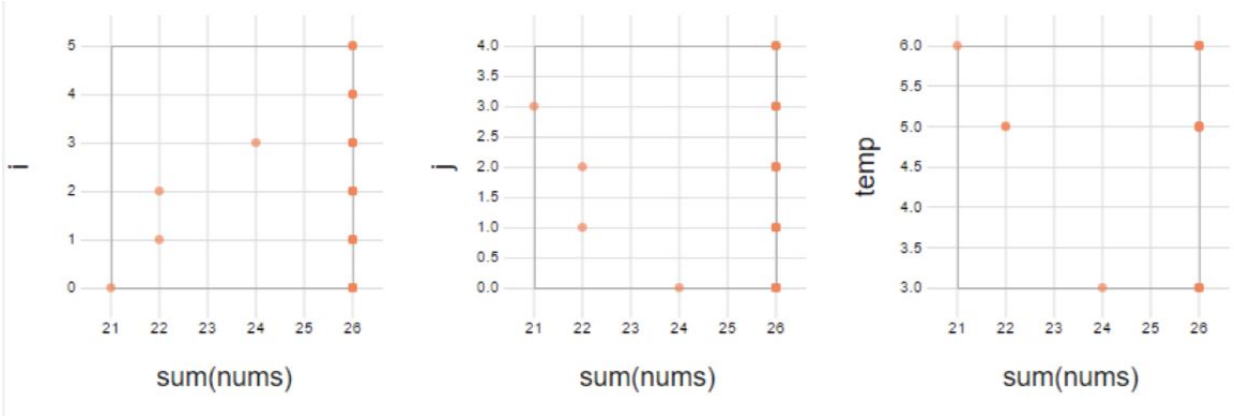

Figure 1. The scatterplots output by Omnicode (Kang, 2017)

The user is given the ability to filter the ranges that the scatterplots represent, such as the value during a certain index range, along with specifying that only certain parts of the code should be graphed. To graph the data, Omnicode gets the information from the live execution stack and parses it. If a user wants to go from graphs to a specific range back to the graphs showing the whole execution stack, then the user only has to un-highlight the selected code. The goal for giving the user the ability to show only selected ranges is to help eliminate the need for temporary variables that are used to evaluate and to make the code easier to read. When a user highlights a portion of the code, all of the scatterplots get updated to show only how the data changes during that code range. If the user wants to only show the scatterplots for a certain variable, then all they have to do is highlight the variable name and then the list of graphs will be filtered accordingly.

2.1.1 Evaluation of the Scatterplots

When the user selects different ranges for the graphs to represent, all of the graphs then get updated to portray the data over that selected range. The user does not have the opportunity to only have specific graphs update, so if they wanted to have only one graph have a different range than the others, they would have to either select the entire range that contains all of the requested information or select the specific range and take a screenshot of the graph to compare with. If the user selects the entire range, then all of the graphs would include all of the selected data points, which the user may or may not want to see. If the user does not want to see all of that information, then the graphs would be cluttered with irrelevant information that could end up confusing or frustrating the novice.

2.2 Overlay

The overlay in Omnicode is shown when the user hovers over a point within their program. The overlay shows the values that a variable will contain at that point, allowing the user to see that information. The information is split into two different columns with Frames being on the left and Objects being on the right. Included in the Frames column are the individual variables for the function the user is hovering over. In the Objects column, the values for different objects, such as arrays, are drawn out with the indexes to help the user see what values are at specific indexes.

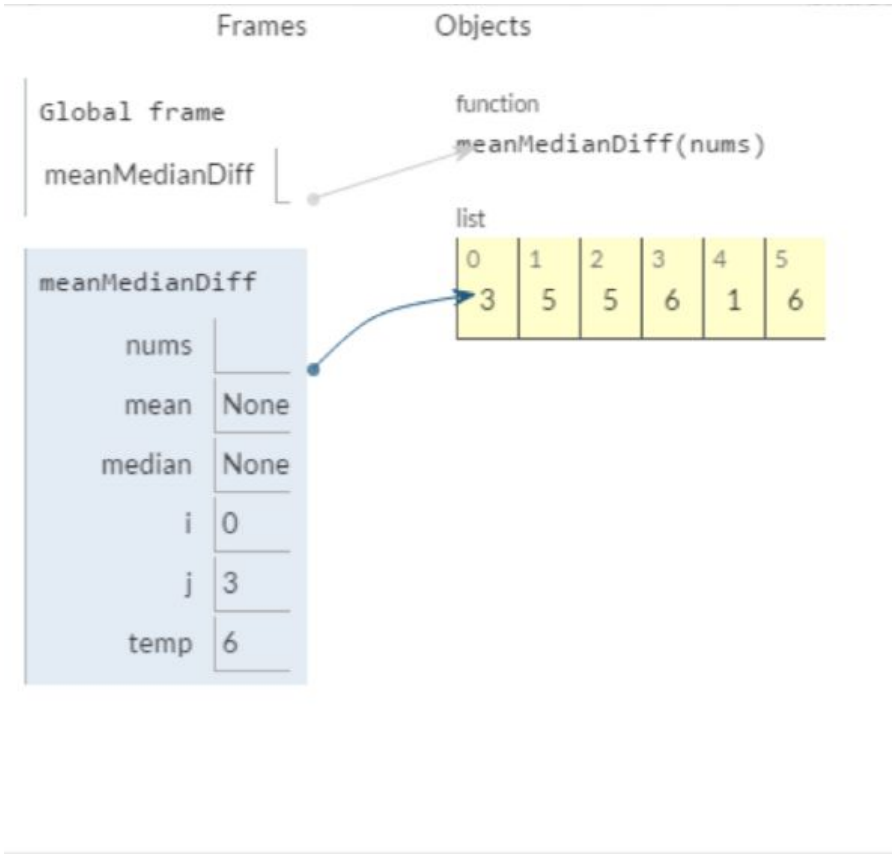

Figure 2. The overlay shown by Omnicode (Kang, 2017)

2.2.1 Evaluation of the Overlay

When the user selects the line of code that they want the popup to show the values for, the popup appears directly underneath that line. The popup covers the code underneath and the user has to unselect the line in order to see that covered code. This would be inconvenient for a novice who wanted to see the specific values at a point but then work through the rest of the function. If the user selects a line of code that appears in a loop, the overlay does not show the values at all of the different iterations of the loop. It only shows one value for that variable in the overlay, but if a graph has been created for that variable then the user could get that information from that panel. There is also no option to select an iteration of the for loop to show the value for a variable within the loop.

2.3 Evaluation of Omnicode

Omnicode is a great tool for helping novices learn to visualize what their program is doing at any specific point in the code. The live data allows the novices to see what is happening without having to hit run, allowing them to not be looking at older data. The user is then able to change a

value of a variable at any point and see how that affects the rest of the execution. The user however is not able to compare that data with older variations of the code. If the user wanted to try out different variations of a statement, then they can only see the current results. From a novice's perspective, it might be beneficial to allow the user to be able to see a previous result that they could mark so they have the ability to visualize the starting point to what they are currently doing. Additionally, if there was a separate pane for the overlay information to be shown, this would make keeping past snapshots easier and would not compromise the user's ability to see the rest of their code. Lastly, if there was the ability to apply filters to individual graphs instead of to every graph, then the user could have more of a freedom to view and play around with the data at their whims instead of having to parse through a more convoluted graph. (Kang, 2017)

3. BlueFix

BlueFix is a tool that exists as an extension for the BlueJ IDE and was developed by researchers at the University of Durham. It's main goal is to take the compiler messages that are returned when errors are found in the code and give an easier to understand alternative that are based on popular vote. When a user gets one of these alternative messages, they are able to vote and comment on it to rank the messages in its helpfulness. Additionally, if a user makes a change to a program and they still get the same compiler message, BlueFix has an algorithm that will change the message to make it easier to understand and even will propose a fix if the user gets the same error for long enough. The main parts for evaluation will be the popular vote mechanic and the alternative messages with a possible fix.

3.1 Popular Vote Feedback

The popular vote system first compares the code that BlueFix thinks is the problem with a list of potential fixes. Once a possible fix is found, it ranks the possible messages based on popular vote, repair success rate, and similarity to the user's code. BlueFix then opens a window that contains the message. Once the user fixes that issue, they are then given the opportunity to vote and comment on how that message helped them solve their issues.

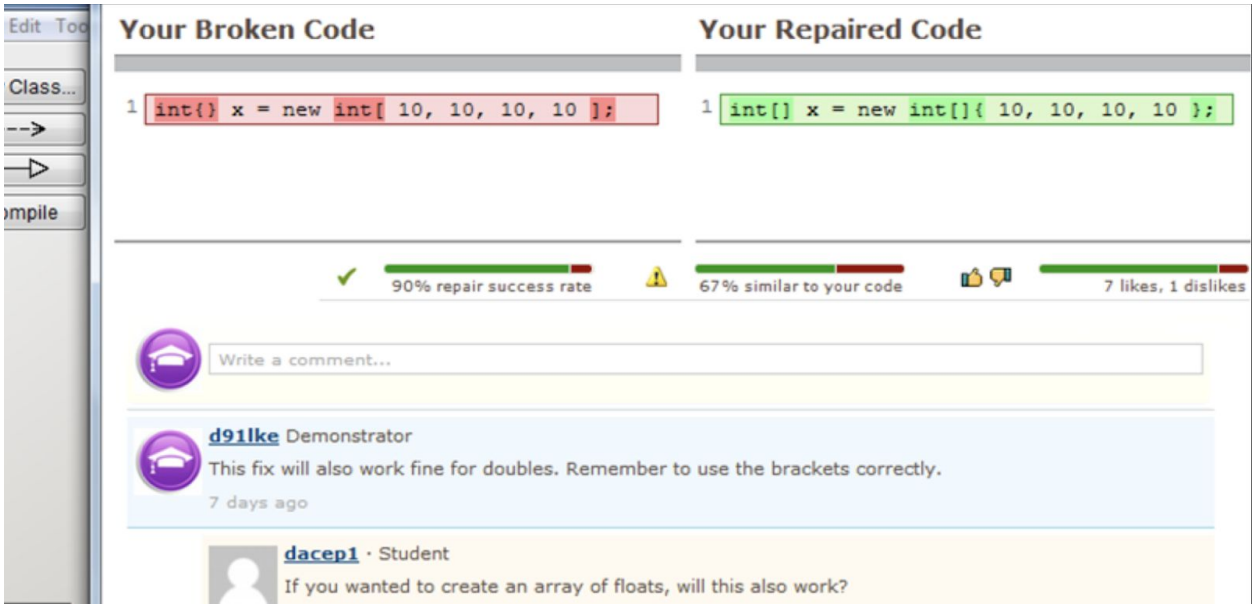

Figure 3. The feedback window shown by BlueFix (Watson, 2012)

3.1.1 Evaluation of Popular Vote Feedback

If a user's code is similar to some of the popular feedback messages but the issue is different than what the messages propose is the fix, the user could become frustrated and convinced that they do not understand what they are supposed to do. Additionally, the user does not have the ability to select different messages, preventing them from exploring and finding the answer that makes the most sense to them. When the user has a question about a message, they are given the ability to comment on a message, allowing others to respond. This allows for misinformation to be spread as a result, along with students not receiving the answers to their questions until someone responds. Since the time and quality of a response are not known, it might be more beneficial for the novice user to be presented with a link to documentation to see if that resolves the question first. Additionally, if the commenting fact was limited to a classroom set, then the teacher or other students in the class could be given a dashboard with easy access and allow for correct and timely information to be given in response.

3.2 Alternative Messages

If the user runs the code again and there was a change made, BlueFix will search its database again for a different message that the algorithm decides would be easier to understand when compared to the last one. Eventually, BlueFix will suggest a potential fix with a message that describes how the fix could help eliminate the error.

3.2.1 Evaluation of Alternative Messages

The user could exploit the system by making a series of small changes that were substantial enough for BlueFix to classify it as a change until BlueFix eventually gave the students the solution. Additionally, BlueFix will not always find the correct issue and might give the students a "solution" to their problem that does not end up fixing the actual error. If BlueFix gave the students a list of possible solutions, but also had a system to try and stop the possible exploits, then the usage of alternative messages could be more beneficial.

3.3 Evaluation of BlueFix

BlueFix is an interesting tool that incorporates discussion and popular voting into giving students feedback to help them learn. The discussion features allows students to ask questions about why a fix worked or on similar topics. The popularity feature allows the messages that have been proved to be helpful to be the first messages that a novice could get when experiencing a problem to allow for quality control. The adapting messages that the novice would receive when working on a problem could help them in the future as they are able to start interpreting what the original error messages mean. However, this could be easily exploited by students who just want to get the answers and not work through the problem to learn and develop their debugging skills. If BlueFix brought up a list of potential solutions and then allowed the novice to browse through those messages and choose for themselves which one fits their problem, then there would be no concern for exploitation. (Watson, 2012)

4. Alternative Solution

An alternative solution that combines the strengths of these two tools to create an environment that would best help novice programmers learn. This tool would be a live IDE similar to Omnicode with the discussion capabilities of BlueFix. This would allow the student to be able to help visualize their code better along with asking for help and exploring other topics that might be of interest. The IDE would have a panel where the overlay information from Omnicode would be displayed. The panel would additionally have more abilities to save snapshots and display the values of a variable at specific points in a loop, without blocking the user's view of the code. The scatterplots for the variables that exist at the time of the selected statement could also exist in this panel, allowing for the user to visualize the entire execution stack up to that point.

The IDE would also have a panel that shows up when the user hits run that contains the alternative messages. The user would be presented with a new version of the error message that was altered to help them understand the issue, along with links to discussions for how to possibly fix that error. The student could then sort through the list and post their questions. The IDE

would work as Omnicode does in the teacher creating an environment that the class would join, allowing the student's questions to be limited to the classroom environment to ensure quality and timely answers. The users could then vote on answers and those messages would only show up if the code passes a threshold for a similarity check through an abstract syntax tree and helpfulness percentage.

5. Conclusion

Novice programmers require the usage of tools that allow them to break down the problem and ask questions. These tools such as Omnicode and Bluefix are great for these cases, however there are some usability concerns from a novice's viewpoint. Examples of these concerns are how information is relayed in Omnicode and how possible fixes are presented in BlueFix. Combining the strengths of these two tools, an alternative solution was presented to help novices get the most out of their early programming experiences, visualize information, and learn how to code.

Works Referenced

Kang, Hyeonsu, and Philip J. Guo. “Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations” *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST 17*, 2017.

Watson, Christopher, Frederick W. B. Li, and Jamie L. Godwin. “BlueFix: Using Crowd-Sourced Feedback to Support Programming Students in Error Diagnosis and Repair.” *Advances in Web-Based Learning - ICWL 2012 Lecture Notes in Computer Science*, 2012, 228–39.