Technical Paper Code Watcher

April 14, 2025

1 Development of the User-Behavior Analytics (UBA) Plugin

We developed the UBA plugin as a VSCode extension[1] in JavaScript using Node.JS. UBA runs in parallel with CGTs without impacting its functionality. It employs event listeners and commands to intercept and/or read information from the document and event objects. This means UBA captures keystrokes and document changes in real time and generates a log file with JSON objects for each newly opened/created file in VSCcode. The UBA plugin captures events. Each event is one of the following types: Start, End, Insertion, Deletion, Focus, Unfocus, Copy, and Paste. For each event, UBA captures some of the following four event properties: (1) Type, (2) Time, (3) Text and (4) Line. Type records which of the 8 events was captured. Time represents the timestamp of when there was an increase in the document text larger than 3 character at a time. Text corresponds to the inserted text. Line is the text of the line where the insertion occurred. Below we describe each event that UBA captures. We also explain the Time/Text/Line corresponding to each event type. When some of these are omitted, it indicates that nothing is captured for that event property:

- **Start**: Records the time that the programmer begins a task in a new document.
  - **Time:** Timestamp of when a new document is in the active text editor.
- **End**: Records the time that the programmer ends a task.
  - **Time:** Timestamp of when a document is no longer open in the active text editor.
- **Insertion**: Occurs when there is an increase of more than three characters in the document text.
  - **Time:** Timestamp of when there was an increase in the document text larger than 3 characters.

[1] The UBA plugin will be made publicly available if the paper is accepted.

  - **Text:** The text that was inserted.
  - **Line:** The text of the line where the insertion occurred.

- **Deletion**: Occurs whenever there is a reduction in the document text, indicating that the programmer has chosen to remove previously written or accepted code. This event can happen by using the backspace key to delete one character at a time, or by selecting and then using backspace to remove a portion or the entire section of code at once.
  - **Time:** Timestamp of when there was a decrease in document text.
  - **Text:** The text that was deleted.
  - **Line:** The text of the line where the deletion occurred.

- **Focus**: Flags when the programmer switched to the IDE from some other application.
  - **Time:** Timestamp when the Active Text Editor moved in focus.

- **Unfocus**: Flags when the programmer switched away from the IDE to another application.
  - **Time:** Timestamp when the Active Text Editor moved out of focus.

- **Copy**: Occurs when the programmer uses the *CMD+C/CTRL+C* key combination anywhere within the IDE. Note that this does not necessarily mean that the copied text will be pasted later on.
  - **Time:** Timestamp when the CMD+C/CTRL+C combination was pressed.
  - **Text:** The text that was copied.

- **Paste**: Occurs when the programmer uses the *CMD+V/CTRL+V* key combination anywhere within the IDE. The next following *Insertion* event contains the suspected pasted text.
  - **Time:** Timestamp when the CMD+V/CTRL+V combination was pressed.

1.1 Post Processing

During post-processing, we quantitatively analyzed user event logs to identify and categorize the users interactions. We defined and compute the user interactions variables from the event logs as follows in Table 1. We took inspiration from the concept of code clones [1] to measure the CGT's suggestions that are retained by the users. Code cloning refers to the process of duplicating a source code during reuse [1]. The purpose of a code clone is to copy-paste programs instead of implementing them from scratch, in an attempt to increase productivity [2]. In the context of an LLM-based code generator, code cloning occurs

Table 1: User Interaction Variables with Derivation Approach

	Category	Description	Derivation Approach
CGI	**Complete generated insertion**	CGT offers a suggestion before the participant begins writing, and the participant accepts the suggestion in its entirety.	Filter for *insertion* objects likely to contain text generated by CGT, ensure that *line* contained no additional text preceding *insertion*
PGI	**Partially generated insertion**	CGT offers a suggestion after the participant begins writing, and the participant accepts the suggestion to finish what they have already written.	Filter for *Insertion* objects likely to contain text generated by CGT, ensure that *line* contained additional text preceding *insertion*
CSLD	**Consecutive single letter deletion**	The participant uses the backspace key to erase a word, letter by letter. If multiple single letter deletions occur within the same second or within a one second difference, it is counted as a CSLD.	Group *deletion* objects where text is one letter with timestamp difference ¡= 1 second
PSLD	**Partial single letter deletion**	The participant uses the backspace key to erase just one letter.	Isolated single letter deletion object
CD	**Complete deletion**	The participant highlights and deletes the entire code, leaving nothing but a newline, carriage return, or space.	Filter for deletion objects where line property is empty
PD	**Partial deletion**	The participant highlights and deletes only a portion of the code, leaving text on the line	Filter for deletion objects where line property is not empty
F	**Focus**	The participant shifted focus to the IDE/ programming task and CGT.	*focus* object
UF	**Unfocus**	The participant shifted focus away from the programming task and CGT.	*unfocus* object
ES	**External source**	The participant utilizes outside sources to add to their code, without the help of the CGT.	*unfocus* object followed by a *paste* object
IS	**Internal source**	The participant copies and pastes code from within the IDE they are writing in.	*copy* object followed by a *paste* and *insertion*, where text property on *insertion* object corresponds to text property on *copy* object
C	**Copy**	The participant copies a portion of the code internally	*copy* object

when we note a line in the final document that matches a line in the log entry file either exactly or almost identically, based on a chosen threshold. To do this, we created a Python script to analyze the Insertion, Copy, and Paste UBA events (since only texts inserted via accepted suggestions or copy-pasting have the potential to make it to the final file). The script works by taking the lists of events from the log entry file and categorizing them based on their type. Then, it filters out invalid insertions based on their relation to the copy and paste events. To avoid double counting, an insertion is deemed valid if it does not match any preceding copy event with the same text and if it does not occur within one millisecond of a paste event. After that, the script takes the filtered insertions and extracts the individual lines of text from the "text" attribute. Once the lines are combined into a single list, the script matches lines from the filtered insertions to lines in the final Python code submitted by the participants. The line-matching algorithm is described below. It uses Levenshtein distance to compute similarity and uses a threshold to compare lines. A threshold of 100 would result in far fewer matches since the program would be looking for identical matches, while a threshold of 50 would most likely return the majority of the lines. Since we want to ensure that only identical to very close matches are accepted, we were not too strict about excluding minor, inconsequential differences, such as newlines and spaces, but also not too lenient about including suggestions that have been significantly modified. Hence, we used a threshold of 95%.

References

[1] Q. U. Ain, W. H. Butt, M. W. Anwar, F. Azam, and B. Maqbool, "A systematic review on code clone detection," *IEEE Access*, vol. 7, pp. 86 121–86 144, 2019.

[2] N. Saini, S. Singh, and Suman, "Code clones: Detection and management," *Procedia Computer Science*, vol. 132, p. 718–727, 2018.

Algorithm 1 Clone Detection

Input: filtered_insertion_lines, document_lines, threshold
Output: matched_lines, unmatched_lines_document, unmatched_lines_insertions
Initialize matched_lines as an empty list
Initialize unmatched_lines_document as a copy of document_lines
Initialize unmatched_lines_insertions as a copy of filtered_insertion_lines
Initialize used_document_lines as an empty set
Initialize used_insertion_lines as an empty set
for each doc_line in document_lines **do**
  Set best_match to None
  Set best_match_score to 0
  Set best_match_index to None
  for each ins_line in filtered_insertion_lines **do**
    if ins_line is not in used_insertion_lines **then**
      Calculate similarity between doc_line and ins_line
      if similarity ¿= threshold and similarity ¿ best_match_score **then**
        Set best_match to ins_line
        Set best_match_score to similarity
        Set best_match_index to the index of ins_line
      end if
    end if
  end for
  if best_match is found **then**
    Append (doc_line, best_match) to matched_lines
    Add doc_line index to used_document_lines
    Add best_match_index to used_insertion_lines
    Remove doc_line from unmatched_lines_document
    Remove best_match from unmatched_lines_insertions
  end if
end for
Return matched_lines, unmatched_lines_document, unmatched_lines_insertions